Student #1, Sprint 4: Analysis report

Group: C1.04.14

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

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	25/05/2023	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

D02:

Requirement nº4 Lecturer: There is a new project-specific role called lecturer, which has the following profile data: alma mater (not blank, shorter than 76 characters), a résumé (not blank, shorter than 101 characters), list of qualifications (not blank, shorter than 101 characters), and an optional link with further information.

Decisions:

- One decision to be made is how to implement the list of qualifications, which we could implement as a list of string or a long single string.

- Another one is the relation between Lecturer and both Lectures and Courses.

Conclusions: Through a discussion in class with the teacher and the other groups we have determined that:

- Given the restrictions (not blank, shorter than 101 characters) the easiest method was to implement it with a single String.

- The best way to implement the relations will be a ManyToOne from both Course and Lecture to Lecturer.

Requirement nº5 Course: A course aggregates several lectures by the same lecturer. The system must store the following data about them: a code (pattern "[A-Z]{1,3} [0-9]{3}", not blank, unique), a title (not blank, shorter than 76 characters), an abstract (not blank, shorter than 101 characters), an indication on whether it can be considered a theory course or a hands-on course (depending on the lectures that it aggregates), a retail price (positive or nought), and an optional link with further information. Purely theoretical courses must be rejected by the system.

Decisions:

- To indicate if a course is hands-on or theoretical, i will implement a function in the service that will calculate the number of each type of lecture. If theory>hands-on, the course will be a theory course, and if hands-on>theory, then it will be hands-on. There is also the third possibility that theory=hands-on.

- The type of course is a derived attribute as it depends on the number of theory and hands-on lectures. It can be persistent or not.

Conclusions:

- As discussed with our teacher in class, I have decided that in case there is the exact number of theory and hands-on courses, a third choice will be added in the enumerated LectureType in case the third option occurs. This option will be "balanced".

- Again, as discussed with our teacher the easiest method to implement derived attributes in most cases will be to make them non persistent.

Requirement nº6 Lecture: A lecture is a document that a lecturer uses to get some knowledge across. The system must store the following data about them: a title (not blank, shorter than 76 characters), an abstract (not blank, shorter than 101 characters), an estimated learning time (in hours, positive, not nought), a body (not blank, shorter than 101 characters), an indication on whether it can be considered theoretical or hands-on, and an optional link with further information.

Decisions:

- To implement Lecture I have made many decisions. Firstly for the attribute lectureType, I have decided to implement it as an Enumerated with three values: theory, handson and balanced. The balanced option is not specified in the requirements, but it will be used when a lecture has the same number of theoretical and hands-on lectures. This decision was approved by the teacher in class.

- The estimated learning time, as it is an estimate and does not need to be calculated between two periods of time, I will implement it as a Double to make it easier to work on. Again this decision has been reviewed by our teacher in a follow up session.

Relation between Requirement nº6 & nº5: In order to implement the bidirectional relation between Course and Lecture i have decided to make a third class called CourseLecture, wich will have a ManyToOne from itself to both Course and Lecture, keeping the id's of both. This decision was reviewed by our teacher in laboratory sessions.

Requirement nº7 LecturerDashboard: The system must handle lecturer dashboards with the following data: total number of theory and hands-on lectures; average, deviation, minimum, and maximum learning time of the lectures; average, deviation, minimum, and maximum learning time of the courses.

Decisions: For the statistics of both lectures and courses (min, max, deviation, average), we have various options. Make each statistic an individual propertia of the class LecturerDashboards or, as suggested by our teacher, make two Maps<String, Date> one for lectures and the other for courses, where the string will contain the type of statistic (max, min, deviation), and the Date the value of the learning time of each statistic.

Decision: I will implement the second option, as it is more elegant and was suggested by our teacher in class.

Requirement nº8: It’s just to produce data to test the application. No analysis is required.

D03:

Requirement nº11 as Anonymous principal: Sign up to the system and become a lecturer.

Decisions: A simple requirement that doesn't require any major decisions to be made.

Conclusions: A button has been added to register as lecturer by entering the corresponding data.

Requirement nº12 as Lecturer: Update their profiles.

Decisions: Again, no mayor had to be made.

Conclusions: The lecturers can now access their data and modify and update it if needed.

Requirement nº13 as Any Principal on courses:

- List the courses in the system that are published.
- Show the details of the courses that they can list (excepting their lectures).

Decisions: This is a simple listing and show of the published courses. The only real decision to be made was how to calculate the CourseType of each course, as it is a derived attribute.

Conclusions: I made an option in the main menu bar that redirects the user (any principal) to the list of courses, from which you can access the individual course data. To calculate the CourseType i have made a function in the Show Service that will get said type according to the number of each LectureType within its associated lectures. This function will be reused for other features.

Requirement nº14 as Lecturer on courses:

- List the courses that they have created.
- Show the details of their courses.
- Create, update, or delete their courses. Courses can be updated or deleted as long as they have not been published. For a course to be published, all of its lectures must have been published.

Decisions: This was by far the most complex requirement to implement.

The list and show were easy to implement. In the menu bar you can access the list of your courses as long as you are logged in as a lecturer. From the list you can access the courses show views.

In the list view there is a button to create a course. Both course Code and CourseType are generated in the service automatically so there is no need to add them as fields on the form.

In the show view, if the course is not published you will be able ti update, delete, add lecture, remove lecture and publish. You will also always be able to see the list of lectures that are aggregated to the course.

For delete, if a course with lectures associated is deleted, the relation and course will be deleted, but not the lecture, that is to say, the CourseLectures will be deleted where it has said course.

To add lectures, I made a form which lets the lecturer select between the published lectures. This creates a new CourseLecture with the course you where in, and the selected lecture.

To remove lecture, the easiest way i could come up with to implement it is to add a button that takes you to a list view of the lectures of the course, go to the view of said lecture, and then, now that we have both the id of the course and lecture, in the show a button will allow the lecturer to remove the lecture.

Finally the publish button will only appear in the show view of the course if the course has lectures (no need to check if they are published as you can only add published lectures). Once a course is published you won't be able to do any operations on the course, with the exception of listing the lectures of said course.

Conclusions: All these decisions were checked and discussed with our teacher during practice sessions.

Requirement nº15 as Lecturer on Lectures:

- List the lectures in their courses.
- Show the details of their lectures.
- Create and publish a lecture.
- Update or delete a lecture as long as it is not published.

Decisions:

for the urls, this post was used as a guide:

https://ev.us.es/webapps/discussionboard/do/message?action=list_messages&course_id=_63009_1&nav=discussion_board&conf_id=_303964_1&forum_id=_206215_1&message_id=_363498_1

As explained here, even though the requirement only asks for the listing of lectures on my courses, I have made both that, and the listing of all my courses. To access the listing of the lectures on courses first you have to access said course. To access all you can do it through the menu bar, both as long as you are registered as a lecturer.

The show and create are both simple enough. The create button was added to the listing of all my lecture views. You can also publish a lecture through a button in the show view, as long as it isn't already published, in other words: The lecture's draftmode has value = true.

Again the update and delete buttons are available on the show view of a lecture, as long as it's draftmode has value= true, as asked in the requirements.

Conclusions: This requirement is time consuming but not that difficult to implement. It's difficulties come with its relation with the former requirement, given the relation of aggregation that have course with lecture.

Requirement nº16 as Lecturer on lecturer Dashboards : Show their lecturer dashboards.

Decisions: As explained in the last deliverable i have decided to implement this using maps to store the data of the statistics of courses and lectures, all of which were calculated in the service.

Conclusions: No real decisions to be made, a simple feature to implement.

D04:

Requirement nº19 : Produce a test suite for Requirements #14 and #15.

Decisions: No analysis is required to implement this requirement.

Requirement nº20: Produce assorted testing data for your test suite.

Decisions: No analysis is required to implement this requirement.

Conclusion

This deliverable, while not complex, was time consuming, mainly because of the assorted testing data to produce previous to the tests. However it was also useful, since during the process of the creation of tests I was able to realize many mistakes in my implementations and was able to improve them.

Bibliography

Intentionally blank.