



Implementing a New Graphical User Interface For GameCourse - GameUI

Patrícia Isabel Dias da Silva

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisors: Prof. Daniel Jorge Viegas Gonçalves
Prof. Sandra Pereira Gama

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. Rui Filipe Fernandes Prada

January 2021

Acknowledgments

I want to thank my mother and my boyfriend for all the support during these thought years, always being there for me, cheering me up on the darkest moments, and always encouraging me to do more and my absolute best. Without your support these years away from both of you and home, this project and life experience would not be possible. I know I am not easy to deal with in those dark times due to my anxiety and stress, so thank you for everything and always believing in me. I love you. I would also like to thank my sister, my stepfather, and grandmother for the support and help throughout all these years.

A special thanks to all the friends that these years brought. I am sure our friendship will last; you are the best I take from this time. Thank you for the support and help on all the tests, exams, and projects, especially the last ones. You help shape the person I am today and will always have a special place in my heart.

I want to thank Imaginary Could for the support through the last year and the possibility of working for them while finishing my master's degree. I am very grateful to all of you. A special thanks to Eva, Bruno, and Paulo as this Thesis would not be the same without your support. An even more special thanks to Rita that helped me during both design and development so I could create the absolute best version of this project, for being such a great friend, for making me company during the late hours of programming and writing, for everything. Thank you.

I would like to acknowledge my dissertation supervisor Prof. Daniel Jorge Viegas Gonçalves, for his insight, support, and sharing of knowledge that has made this Thesis possible. I also would like to thank Tomás Alves and Amir Nabizadeh for the support during each phase of this Thesis and for making sure my English and writing are on point. Finally, thanks to that FCT partially supported this work through project GameCourse PTDC/CCI-CIF/30754/2017

Last but not least, to all my friends and colleagues that made time to participate in the several user tests. Thank you. To all the friends back home that life showed to be the best I could have, thank you for the support, thank you for always being there even though time was not in our favor.

To each and every one of you – Thank you.

Abstract

Education is taking a step forward into the new technologies. A Learning Management System (LMS) is a system that handles all the process related to learning, through which a teacher can create a course and manage the activities related to it. Like any other system, its User Interface (UI) impacts its acceptance, which means that although it may have well implemented features, it can be rejected if the user is not able to understand what he/she can do with the system and how. GameCourse is a web-based LMS used at the Multimedia Content Production course at Instituto Superior Técnico, that uses a basic and minimalist UI where users get confused and lost while performing simple tasks since they feel like there is nothing there to help them. The purpose of GameUI is to create a user-friendly UI for this system, making it clear, consistent, easy to use for either experts or beginners, so the user can take full advantage of the features of the system.

Keywords

Learning Management System; User Interface; Usability; User Experience; GUI.

Resumo

A educação está a dar um passo à frente nas novas tecnologias. Um Sistema de aprendizagem online é um sistema que trata de todos os processos relacionados com a aprendizagem, através do qual um professor pode criar um curso e gerir as suas atividades. Como qualquer outro sistema, a sua interface de utilizador (IU) afeta a sua aceitação, isto significa que, embora as funcionalidades possam estar muito bem implementadas, o sistema pode ser rejeitado se o utilizador não conseguir entender o que pode fazer com o mesmo. O GameCourse tem sido o sistema de aprendizagem online utilizado no curso de Produção de Conteúdo Multimédia do Instituto Superior Técnico nos últimos dez anos. Apesar de usar uma IU simples e minimalista, os utilizadores ficam confusos e perdidos enquanto realizam tarefas simples. O objetivo deste projeto, GameUI, é criar uma interface de utilizador para este sistema, tornando-o, claro, consistente e fácil de usar para especialistas ou iniciantes, para que o utilizador possa aproveitar ao máximo os recursos do sistema.

Palavras Chave

Sistema de aprendizagem online; Interface; Usabilidade; Experiência do utilizador; Interface Gráfica.

Contents

1	Introduction	1
1.1	Objectives	2
1.2	Organization of the Document	2
2	Related Work	3
2.1	Usability and User Experience comparison	3
2.2	GUI elements analysis	8
2.2.1	Structure	9
2.2.2	Initialization	11
2.2.3	Customization	16
2.3	Discussion	19
3	The GameCourse system	24
3.1	How it works	24
3.2	Current UI	25
3.3	Preliminary work with User Tests	27
3.3.1	Results	27
3.3.2	Requirements	30
4	GUI Design	31
4.1	Design Process	31
4.1.1	Modals	32
4.1.2	Menus	32
4.1.3	Listing pages	33
4.1.4	Roles page	35
4.1.5	Login page	36
4.1.6	Documentation	36
4.1.7	View editor	36
4.2	Low Fidelity Prototype	37
4.3	Heuristic Evaluation	38
4.3.1	Participants	38
4.3.2	Procedure	39

4.3.3	Results	39
4.4	Discussion	40
5	Development	41
5.1	Structure and style of main components	41
5.2	New GameCourse structure	42
5.2.1	Code refactoring	42
5.3	Navbar	43
5.4	Courses page	44
5.4.1	Class preparation	44
5.4.2	Admin version	45
5.4.3	Search, filter and order algorithm	47
5.4.4	Non admin version	48
5.5	Users page system and course domain	49
5.5.1	Class preparation	49
5.5.2	Users page - system domain	50
5.5.3	Users page - course domain	51
5.6	Settings page	53
5.7	Roles page	53
5.7.1	Undo/Redo mechanism & State Manager	55
5.8	Modules pages	56
5.9	New Configuration System for Modules	57
5.9.1	Code Restructure	57
5.9.2	Configuration page	57
5.9.3	Documentation	60
5.9.4	Badges configuration page	60
5.10	Views page	61
5.10.1	View Editor	62
5.11	Documentation Pages	65
5.12	Changes to complement new functionalities	66
5.12.1	Plugin module configuration page	66
5.12.2	Autentication	67
5.12.3	Import and Export	68
5.12.4	Autocomplete	69
5.13	Main page	69
5.14	My Information	70

6 Evaluation	71
6.1 User Tests	71
6.1.1 Participants	71
6.1.2 Procedure	71
6.2 Results	72
6.2.1 NasaTLX	75
6.2.2 System Usability Scale	75
6.2.3 User Experience Questionnaire	76
6.3 Discussion	76
7 Conclusion	78
7.1 Future Work	79
A List of the 64 LMSs	83
B Websites with top X LMS lists	84
C First User tests	85
C.1 Introduction	85
C.1.1 EN Version	85
C.1.2 PT Version	85
C.2 List of Tasks	85
C.3 Interview	86
C.3.1 EN Version	86
C.3.2 PT Version	87
D GameCourse Use Cases	88
E Heuristic Evaluation Results	90
F Documentation section about Modules configuration	92
F.0.1 Module Configuration	92
F.0.1.A General Inputs Section	92
F.0.1.B Listing Items Section	93
F.0.1.C Personalized Section	95
G List of tasks available on the LFP	96
H List of tasks for the final User Tests	97
H.1 Old Version	97
H.2 New Version	98
I Schema of GameCourse database	99
I.1 Old Version	99
I.2 New Version	99

List of Figures

2.1	Edmodo's home page.	9
2.2	Selection of the wireframe on SapLitmos.	9
2.3	Schoology navigation bar and its second layer.	10
2.4	Menu inside Canvas course.	11
2.5	TalentLMS possible tasks.	11
2.6	Layers of sliders on Blackboard.	11
2.7	Menu path on TalentLMS.	11
2.8	Create new course on Moodle.	12
2.9	Create a new course on Edmodo.	12
2.10	Change roles and views on Canvas.	13
2.11	Change roles and views on iSpring.	13
2.12	Change roles and views on TalentLMS.	13
2.13	Moodle's role assignment.	14
2.14	Creating a user type on TalentLMS.	14
2.15	Add user one by one - Edmodo.	15
2.16	Add users by ID - Canvas.	15
2.17	Profile page seen by self.	16
2.18	Profile seen through listing page.	16
2.19	Top of the profile page on TalentLMS	16
2.20	Extra features on SapLitmos.	17
2.21	Points on TalentLMS.	17
2.22	Badges assignment on Schoology.	17
2.23	Notification on Moodle.	18
2.24	Notification on Canvas.	18
2.25	Add content of a page on iSpring.	19
2.26	Theme customization on SapLitmos.	20
3.1	Side menu - system domain.	25

3.2	Modules on the settings page.	25
3.3	Navigation bar - system domain.	25
3.4	Navigation bar - course domain.	25
3.5	View editor with table (orange), block (yellow), text (green), image (blue) and edit buttons (red).	26
3.6	No syntax error on expression.	26
3.7	Syntax error on expression.	26
3.8	Documentation tabs.	26
3.9	Text box to insert new user (Teacher).	28
3.10	Middle page to edit page/view.	28
3.11	Change user roles.	28
3.12	Roles hierarchy.	28
3.13	Error that helps the user understand his/her error.	29
3.14	Error with information only a programmer can understand.	29
4.1	Add new modal.	32
4.2	Verification modal.	32
4.3	Navbar using white space as dividers.	33
4.4	Standard icons used.	34
4.5	Courses page for admin users.	34
4.6	Courses page for non admin users.	34
4.7	Views page.	35
4.8	Modules page.	35
4.9	First version of the Roles page.	35
4.10	Final version of the Roles page.	35
4.11	Login page.	36
4.12	Functions list on documentation.	36
4.13	Add a new part in the View Editor.	37
4.14	Change aspect of a part.	37
5.1	Navbar inside a course with open dropdown on settings	43
5.2	Flowchart of the checkNavbarLength algorithm	44
5.3	Courses page for admin users.	45
5.4	Add a new course modal with color picker.	46
5.5	Verification modal before deleting a course.	47
5.6	Flowchart of order mechanism	48
5.7	Courses page for non admin users.	49
5.8	Users page on the system domain.	50

5.9	Edit User modal on the system domain.	51
5.10	Select mode of the add user modal.	53
5.11	Add existing mode of the add user modal.	53
5.12	Roles page with the roles hierarchy.	54
5.13	Modules page inside a course.	56
5.14	Module modal with disable on/off button.	57
5.15	Module modal with enable on/off button.	57
5.16	Available types on general inputs section.	59
5.17	Badges configuration page.	61
5.18	Views page.	62
5.19	Intermediate page for aspect selection.	63
5.20	View Editor with a block on edit layout mode.	63
5.21	Table Part on edit layout mode.	64
5.22	Top part of the Edit Part modal.	64
5.23	Add Part modal.	64
5.24	Switch Part modal.	64
5.25	Search for "image" on the Views page of the documentation	66
5.26	Plugin configuration page.	67
5.27	Login page.	68
5.28	Import system users.	69
5.29	Input with suggestion from the autocomplete.	69
5.30	Content of the Main page with shortcuts.	70
5.31	Content of the My information page.	70
6.1	Mean value and Standard Deviation on each scale of the User Experience Questionnaire.	77
1.1	Complete Schema of GameCourse database on the initial GameCourse.	99
1.2	Complete Schema of GameCourse database on the final version of GameCourse.	100

List of Tables

2.1	Five quality components of usability.	4
2.2	Percentage of students that answered above neutral on the questions about Moodle and BlackBoard (N=53).	4
2.3	Percentage of compliance of the three LMSs regarding each heuristic defined by Nielsen.	5
2.4	Items of interest, their mean values and standard deviation in each system.	7
2.5	Results per category.	8
2.6	Final results from the four studies.	20
2.7	Structure analyse of the eight LMSs.	21
2.8	Initialization analyse of the eight LMSs.	22
2.9	Customization analyse of the eight LMSs.	23
4.1	Page organization on the new GUI.	31
4.2	Levels of severity of a usability problem	38
4.3	Number of problems found by level of severity.	39
4.4	Number of problems found by heuristic as well as their presence in each level of severity.	39
6.1	Success rate of tasks 1 to 10 and 12 per version.	73
6.2	Results for the tasks 1 to 5, 7, 9 and 10. (R - Rejected; NR - Not Rejected;)	73
6.3	Results for the tasks 6, 8 and 12.	73
6.4	Results for the impact of having more inputs to fill.	74
6.5	Results of task 11 separated by the first tested version.	74
6.6	Success rate of the tasks 13 to 17 on the new version.	74
6.7	Results of tasks 13 to 17 on the new version.	75
6.8	Results of the NasaTLX questionnaire.	75
6.9	Results of the SUS questionnaire.	75
6.10	Analysis per system of the results of the UEQ questionnaire.	77

Acronyms

API	Application Program Interface
LMS	Learning Management System
UI	User Interface
UX	User Experience
GUI	Graphical User Interface
SUS	System Usability Scale
Nasa-TLX	NASA Task Load Index
UEQ	User Experience Questionnaire
LFP	Low fidelity Prototype
SM	State Manager

Chapter 1

Introduction

Throughout the years teachers and researchers have been trying to find an efficient way to improve student's performance and engagement in classes. Some say this improvement is possible by keeping the original form of lectures with assignments [1, 2], while others say the key to success is to use new technologies and incorporate them in and out of the classroom. This new approach to teaching started by including interactive whiteboards [3] and clickers [4] to make a more interactive class, continued towards e-learning systems [5, 6] and now it is focus on Learning Management System (LMS).

A LMS is more than a simple online course where students see a pre-recorded version of a class. A LMS [7] is a framework that handles all the process related to learning, through which a teacher can create a course, manage the activities related to it, assign different learning paths to his/her students and gather results and statistics. Moreover, a LMS can even integrate game elements (gamification) and make the learning process more fun and exciting for the students [8, 9]. Applying a LMS in a course can bring a lot of benefits [7]: each student can go at his/her own pace; there is no need to wait for the exam to learn the next subject, as soon as a skill is acquired the student can learn the next one; the teacher knows exactly where each student needs help and can give precise feedback and support.

However, the efficiency of LMS is not only based on the features of the system, and how well it works, it also relies on the User Interface (UI), as it is through it that the user can interact with the system. We can have a perfect system with the perfect features and still have it fail outside the lab if the user does not understand how to use it, like the case of the Opensource LMS Ilias when compared with Moodle and Atutor [10].

For the past ten years, GameCourse has been used on the Multimedia Content Production course from the Msc. of Information Systems and Computer Engineering at IST. GameCourse is a web-based LMS that allows anyone who wants to teach to create and manage their courses. Besides creating a course and adding the corresponding Students and Teachers, we can define multiple roles, define them hierarchically, and associate them with a user. We can also create personalized pages, restrict their access depending on the viewer's role, and have dynamic information also depending on the viewer. Furthermore, it is possible to enable gamification related modules that will make the course more fun and interesting.

Although a lot of changes have been done through the years to improve GameCourse [11, 12], its UI has only had two version and is seen as "old". It gets users confused while performing basic tasks, which may lead them to loose interest on the system. Dourado [11] showed that the system scored a mean of 57.63 for the System Usability Scale (SUS), which is considered a low value, and mean of 28.92 for the NASA Task Load Index (Nasa-TLX), that should be as low as possible. Both these results along the reports from the users show that the system UI needs to be improved.

1.1 Objectives

The objective of this thesis is **to develop a new and more user-friendly** Graphical User Interface (GUI) of the Learning Management System called GameCourse, currently used on the course Multimedia Content Production, in the Computer Engineering Master Degree at Instituto Superior Técnico. This new interface will respect the principles of UI, Usability, User Experience (UX) and make it easier for teachers to create and manage their courses.

1.2 Organization of the Document

This document is divided into seven chapters. Following this chapter, we have Chapter 2 where we compare some of the existing LMSs in terms of Usability and UX, through the analysis of existing papers. As well as in terms of GUI elements through visual analysis and experiment of the same systems. At the end we summarize the current problems and discuss the most used GUI elements for each task. Then in Chapter 3, we describe the initial state of the GameCourse system and analyse its performance through formative User tests done with ten users, that allowed us to know the requirements of the system. Afterwards, we have Chapter 4, through which we explain the design process, the solution created and the Heuristic Evaluation done with experts to find Usability flaws on the solution. In Chapter 5 we describe the implementation of this project and all the new functionalities of the developed system. Chapter 6 covers the evaluation performed on the system with the summative User tests. To end we have Chapter 7 with the conclusion and future work.

Chapter 2

Related Work

There are more and more LMSs every day. At this moment, there are more than 60 LMSs available online to be purchased or used as Open Source (Appendix A). With so many options, it is hard for a teacher to choose which system is the best. That's why studies, that bring up special features and important issues that could influence the teacher's choice are so important, like Nadire Cavu's [13], which compares six different open source LMSs on some selected features, and Tania Acosta's [14], that compares the accessibility on three LMSs, or even Croitoru's [15] research that deeply analysed the features of other five LMSs. Besides those studies we have online platforms that classify these systems in terms of best system, the most used and the ones with best usability results (Appendix B). From those 60 I was able to select the ten most promising LMSs to analyse further in this chapter: Talentlms, SapLitmos and Moodle with 7/8 votes, Canvas with 5/8 votes ,Blackboard with 4/8 votes and Schoology, Edmodo, iSpringlearn, eFront and learnUpon with 3/8 votes.

2.1 Usability and User Experience comparison

A LMS can be analysed in many different ways: by exploring it's features, how well they work and how many it has; the efficiency to which the system responds to the user interaction; the UX while using the system and it's Usability. For a good development of a UI, the usability and UX are the most important analysis [16]. Although these two fields can seem to represent the same thing, usability is defined as "the ease with which a user can learn to operate, prepare inputs for, and interpreter outputs of a system component" [17], whereas UX as the "user's perceptions and responses that result from the use and/or anticipated use of a system, product or service" [18], which means, the way the user feels when interacting with the system.

There are at least three methods that we can use to analyse the usability and UX of a system: Surveys and Questionnaires, Observation, and Heuristic Evaluation. [19] The first two methods require interaction between the user and the system. In the first one the user experiences the system and then is asked questions through an interview or survey, while in the second there is a supervisor recording everything the user does and analysing it later. The third method requires a knowledge of the usability principles, including the ten heuristics defined by Jakob Nielsen [20, 21], one of the most well-known researcher in the field. These heuristics cover the five quality components of usability (Table 2.1) [19] defined by Nielsen and are the key to success when designing an UI.

Even though the majority of the ten selected LMSs are evaluated in terms of usability on the mentioned websites, there is not an official document with that analysis to explore here. For that reason, only the oldest, the most used and education focused systems, like Moodle, Blackboard, Canvas and Schoology are mentioned in the following researches.

Learnability	How easy is for the user the first use of the system.
Efficiency	How fast can a user start completing tasks.
Memorability	How easy is for the user to use the system again without having to relearn it.
Errors	How many errors does the user make and how easily can he recover from them.
Satisfaction	How much does the user enjoy the design of the system.

Table 2.1: Five quality components of usability.

Back in 2007 Machado and Toa [22] made a study using the Surveys and Questionnaires technique where they compared the UX of Moodle, an Open Source LMS, and Blackboard, a propriety LMS. This study was conducted in a school where the BlackBoard system was already implemented, and for that no special extra time was given to the participants to retry this system. Participants were divided in three groups: students that would use Moodle for the next nine weeks in a specific course (group 1); students that had previously done the course using BlackBoard (group 3); and professors that would teach the course using Moodle (group 2). The first group of students was the only one with contact with Moodle, and for that, the only ones answering direct question of both systems. The second group only worked as base of comparison for how students felt towards the BlackBoard system without trying another one. Participants were questioned using surveys, where apart from the open questions at the end, all other items used the Likert scale.

Due to the low number of professors (N=4), they were not able to take reasonable quantitative analysis. However three out of four said that they would rather use Moodle than BlackBoard. From the second group they were able to conclude that the students did not had strong feelings towards BlackBoard and would easily prefer to use another LMS, even though they answered "somewhat agree" when asked if they felt the system was easy to use.

Question	Moodle	BlackBoard
You felt it enhanced the instruction that you received in your current course	64%	55%
You felt it enhanced the organization of learning materials	70%	55%
You felt that communication tools available in it enhanced interaction with your instructor	53%	36%
You felt that the web-based resources available to you in it were effective learning tools	64%	42%

Table 2.2: Percentage of students that answered above neutral on the questions about Moodle and BlackBoard (N=53).

The more interesting results are from the first group where 71% of the students felt that the Moodle system was easier to use than the BlackBoard, and 75% would prefer to use Moodle instead of BlackBoard. This preference can be seen in the following Table 2.2 that shows the amount of students that answered "somewhat agree" or "strongly agree" to the survey questions. The number of students to answer any of the disagreeing levels is very small. However, the neutral level is still very populated, which means that both Moodle and Blackboard still have some usability flaws.

Later in the year 2008 a group of six researchers analysed those flaws of the Moodle system and another two systems, Sakai and dotLRN [23]. They gathered five usability experts to participate in the study and apply the Heuristic evaluation method. The three platforms were prepared with the exact same content and five tasks with no time restriction were given to the experts (in the conclusion of the article, the authors say that these tasks should have been selected more carefully, adding specific situations that tested the heuristics directly, like errors). After spending the needed time in the platforms all the experts filled a data log sheet containing more or less than 200 of the checkpoints of usability measure. If a checkpoint was violated they would have to score the severity of that violation, low, medium, serious or critical.

H	Heuristic	Moodle	Sakai	dotLRN
H1	Visibility of the system status	65%	78%	81%
H2	Match between system and real world	60%	75%	82%
H3	User control and freedom	62%	65%	62%
H4	Consistency and standards	72%	83%	80%
H5	Help users recognize, diagnose and recover from errors	90%	77%	88%
H6	Error preventing	48%	67%	55%
H7	Recognition rather than recall	66%	80%	81%
H8	Flexibility and efficiency of use	22%	30%	44%
H9	Aesthetic and minimalist design	76%	89%	90%
H10	Help and documentation	65%	78%	67%

Table 2.3: Percentage of compliance of the three LMSs regarding each heuristic defined by Nielsen.

Their results were shown in percentage, because the amount of checkpoints per heuristic are not the same. From the three platforms the one with the highest score was dotLRN with 78% of compliance in total, followed by Sakai with 77% and Moodle being far behind with 68%. Even though Moodle is the most used platform, in this study is the one with lowest usability score. Looking closer at every heuristic (Table 2.3) their study shows that H8 is a big problem in all the LMSs selected, which means the systems are not providing accelerators for expert users, making them take more time interacting with it than they would like or need to. H5 and H9 are the two heuristics with better scores being around the 85%, meaning, all the systems have a clear and clean design that does not have more information than it needs to have, together with the ability to respond well to user errors, helping him/her recover from them. In the H3 the three systems get similar results, around 63%, giving the user the ability to do, undo and redo anything he/she wants.

Regarding errors related heuristics, Moodle has an interesting result: even though it has the highest score on H5, H6 (error prevention) is the second lowest score of this system. In another words, Moodle does not prevent users from making mistakes but helps them recover from them in an easy and clear way. Another interesting result from this study is that, although Moodle is the one that breaks more checkpoints, it is the one with less critical flaws, and also critical and severe flaws together. These final results may be the reason why Moodle is still more used than the other LMSs, considering that critical and severe flaws are the reason why users tend to leave and reject a system.

From the first study we concluded that the UX on BlackBoard is lower than in the Moodle, and that users tend to reject this system, but the usability of this first system was not analysed. That was done by Thacker, et al. in 2014 [24]. They compared the usability of two systems BlackBoard and Canvas, but, instead of focusing in the students perspective, they focus on the teacher side. For this study, five teachers were selected and two types of analysis were made. First users were observed and recorded while using a technique called Think-aloud, where users say what is on their mind while using the system. Then after completing five tasks in each system they were asked questions through a survey and interview.

Throughout the observation, they were able to find specific problems of the systems and UIs that made users spend more time on tasks, get frustrated, and give up doing them. One of the topics mentioned several times was the semantic confusion on the menu and labels of the BlackBoard system. The labels "information" and "content" had no apparent difference as also "item" and "assignment", which is related to the H2. Another complaint was related to error reports on the system. The error messages were harsh and not close to the error's origin, which is a clear flaw on H5. This system also turned out to be not very clear in the middle of the process of doing things, flaw on H1, like creating a new quiz, where it was not said anywhere that the questions should be put in the next process page, and many users had to come back and delete them from the first page. The last complaint was about H9, when completing a simple task, many options were shown that confused the user and made him/her lose time.

For Canvas, users complained and got confused if some items were clickable or not, as the system by default makes some items gray until they have content. This flaw goes into the H4, as usually gray items means disable button. When creating a multiple choice quiz, users also found another break at H4: the way to choose the correct option was not clear as in Blackboard that used radio buttons next to the options, a common use for multiple choice questions.

Regarding surveys and interviews, eleven questions were made, using a six-point Likert scale, about each LMS having in mind the ten heuristics of Nielsen. When asked about the Canvas system, results are very positive, having 4/11 questions answered with "agree" 80% of the times and the left seven with 100%. On the other hand, on BlackBoard only 6/11 questions were answered in majority as "agree", one with three votes (60%), one with all and the rest with four votes (80%). The left five questions were answered with "disagree", two of them with four votes and the others with all votes. These five questions are related to the Heuristics H3, H2 and H9, although it was mentioned a flaw in the H5 during the tasks, only one of the users experienced it and that is visible in the research data as the one vote that disagrees on this heuristic. The interviews complemented these results as the users said that the Blackboard system is hard to use and would rather use Canvas than Blackboard in classes.

A more recent study, done in 2016 by Sahid and Santosa [25], also analysed the Moodle system in terms of UX, while comparing it with another previously chosen LMS, Schoology. This comparison was made using the Questionnaires method based on six categories that define UX:

- Attractiveness: general impression of the system;

- Dependability: user feeling in control, secure and if the system is predictable;
- Efficiency: how fast, efficient and organized is the system;
- Novelty: creativity and innovation of the system.
- Perspicuity: easiness and familiarity of the system;
- Stimulation: how interested and willing to use the system is the user;

Data was collected from 38 students, that after using each of the systems in two university courses answered a questionnaire of 26 items. Each item had two sides (ex: annoying and enjoyable) with 7 points for classification, later translated from -3 to +3. Having done a lot of different analysis on the collected data, the most interesting is the means scale (Table 2.4): that allow us to understand the average opinion of the users (mean value), and if their opinions converge (standard deviation - std).

Item	Moodle mean	Moodle std	Schoology mean	Schoology std
2- Understandable or not	1.5	1.2	1.7	1.0
3- Dull or Creative	0.0	1.4	0.9	1.3
6- Exciting or Boring	0.3	0.9	0.6	1.1
10- Conventional or Inventive	0.6	1.4	0.6	1.3
12- Bad or Good	1.2	1.4	2.0	1.0
13- Easy or Complicated	0.9	1.3	1.1	1.6
26- Innovative or Conservative	0.1	1.6	1.4	1.2

Table 2.4: Items of interest, their mean values and standard deviation in each system.

The highest score for Moodle was 1.5, regarding if the system was understandable or not (Perspicuity), with std of 1.2, we can conclude that the majority of the answers were positive and confirm that Moodle is an LMS easy to understand. The worst scores were 0.0 and 0.1, when asked if the system was creative or dull and innovative or conservative, correspondingly. Besides being a mid point value that shows uncertainty from the users on how to classify the system, the std values were 1.4 and 1.6, which shows even more uncertainty. These two lowest points are related to the Novelty category, meaning is not very clear where Moodle stands in terms of innovation and creativity.

For Schoology, the highest score was 2.0 on whether the system was good or bad (Attractiveness), with std of 1.0, means all students agree that the system is good to use and some even gave it the max score. The two lowest scores with 0.6 points were: if the system was exciting and if the system was conventional or intuitive. Although both of them are positive values, their std were 1.1 and 1.3, correspondingly, which translates uncertainty on how to classify Schoology. Like Moodle, Schoology also has one item with 1.6 of std, the difficulty of the system, which may be related to users previous experiences with other LMSs. These three low points of the system, in contrast to Moodle, are not all from the same category, being from Perspicuity, Novelty and Stimulation, which makes the system a more balanced.

It is when we gather all data by categories (Table 2.5) that we can really see the differences between these two systems. Moodle highest points are Perspicuity and Efficiency with values just above the 1.0, while Schoology has only one category below the 1.0, Novelty. Schoology highest points are

Attractiveness and Efficiency with more than 1.5 points. From these analyse, authors concluded that Schoology has a better impact on students. The reason that may be behind the success of Moodle is its customization, that helps making a better job in the UX of the system.

Category	Moodle	Schoology
Attractiveness	1	1.5
Perspiciuity	1.3	1.4
Efficiency	1.2	1.6
Dependability	0.6	1.3
Stimulation	0.9	1.2
Novelty	0.3	0.9

Table 2.5: Results per category.

2.2 GUI elements analysis

The previous mentioned studies analysed the LMSs in terms of usability and UX, that see the user interface as a whole. The following sections present an analysis of the GUI elements used in each part of the UI, regarding the tasks the user has to make. From the top 10 systems, only the first eight were possible to analyse, as the last two required personal permission that was not conceded by the companies. From the eight systems six of them were analysed on their free trial web platform, Moodle was installed in its last version (3.7.2) and Blackboard was analysed through its coursesites web section.

As described before the main objectives of the GameCourse system is for teachers to be able to manage their courses and integrate them with gamification, creating a more interactive environment for students and increasing their engagement on the course. The system has long list of Use Cases (Appendix D), from which three sections of analyse were created:

- Structure:
 - Organization of the home page.
 - How possible tasks are displayed.
 - How the menu path is shown.
- Initialization:
 - Create a new course and edit it.
 - Define possible user types or roles.
 - Change the role through which the user sees the system
 - Add new users.
 - How the profile is organized and how to edit it.
- Customization:
 - Add extra features or modules.
 - Gamification configuration.
 - Notifications.
 - Create a theme and change it.
 - How to create a page or a view.

2.2.1 Structure

The **Home page** is the first official page of a system, after login, and it is usually design to provide basic information about the items it holds and let the user know what tasks can be done. Both Schoology and Edmodo present a page that behaves like a news feed (Figure 2.1), having the latest announcements in the courses, new assignments, events or activities around the platform. However, Edmodo's one is little more difficult to digest due to the amount of information surrounding the center section and lack of color contrast, its design resembles Facebook and Twitter. Schoology also has the possibility of showing another type of organization on the home page. It can display the list of courses of the user, just like TalentLMS, SapLitmos and iSpring (all three in learner mode), Moodle, Blackboard and Canvas. Some of them use cards and grids to present the items, others do it in listing format using cards or links. In particular, Blackboard uses the courses listing page as the home page. Besides the listing, SapLitmos, Schoology and Moodle have a second column for upcoming events, calendar and achievements.

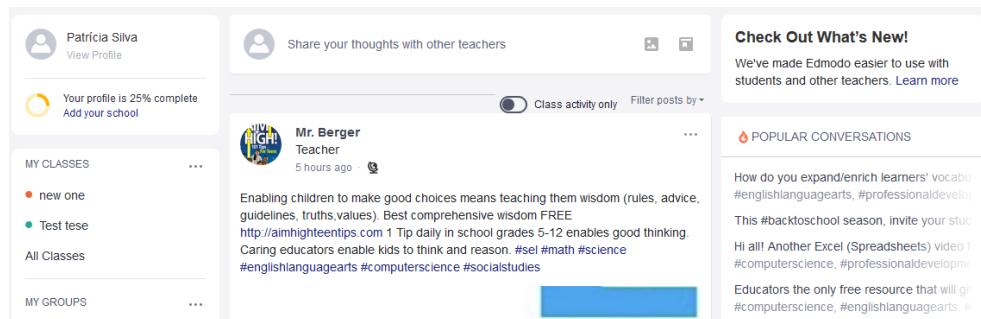


Figure 2.1: Edmodo's home page.

iSpring has two main columns which gather information about the newest content on the courses and the assignments ready to be manually evaluated. On top of those columns, it has a bar with statistics regarding the courses, which can also be seen on the SapLitmos administrator home page and TalentLMS learner and instructor page. Regarding the TalentLMS home page on administrator mode, we have two main sections: one with the possible tasks on the system and another with statistics that can be selected from a toggle button on the top or/and filtered on date in another toggle button.

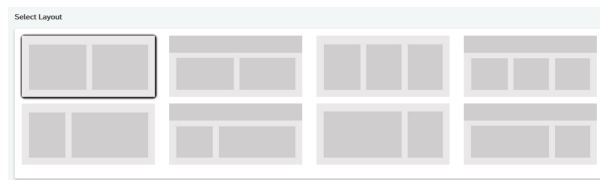


Figure 2.2: Selection of the wireframe on SapLitmos.

The most interesting home page is the one from SapLitmos on administrator mode, as it can be personalized, within some possibilities. By creating or editing a dashboard layout we can give it a name and description to later identify the layout, select the structure of the page from a set of wireframes (Figure 2.2) and then choose which are the most appropriated widgets for each section of the wireframe.

The selection of the widget is done through a modal, with a dropdown list and preview, that appears when selecting a section of the wireframe. As the widgets are being selected a preview is being shown at the same time with the final result of the layout. We can define which of the layouts is used by clicking on a checkbox while editing the layout. This checkbox gets unnoticeable in the middle of two big inputs, description box and wireframe selection. It would be more visible near the save button.

Moodle also let us customize the main page but with more limitations than SapLitmos, as the wireframe is always the same, two columns. When on customizing mode we can drag and drop to change the order of the blocks, add a new block using the button “Add a block” on the menu and by clicking on the settings button, standard icon gear-wheel, we can hide or show a block. Also on this button we can define the permissions of the block which will take us to a new page where we can define who can edit and view the block. Some blocks also let us change the format to display items, how to filter them and the order which they will be shown by.

The **possible tasks** of the system are in the majority of the cases shown in a navigation bar, either on the top or at the left side of the screen. On these bars only the main task or item related is mentioned, like courses and users, using text links that can sometimes be followed by icons. Moodle has the option to completely hide it’s side bar and Canvas to make it smaller by only showing the icons. All of these links lead to the listing page of the item, except from Schoology and Canvas, that for some items show a second menu layer (Figure 2.3) where the user can select a particular item (going to that item page) or select to see all of them (going to the listing page). While Canvas uses text links for that new layer, Schoology uses cards. Besides this second layer, Canvas also has an inside item menu to navigate within a course or profile sections. It is on this menu that we can see an improvement from the flaws detected on the studies presented on section 2.1 [24]. Instead of using the color grey on the text link when there is no content yet, they use a crossed eye icon next to the text link (Figure 2.4).

As mentioned at the start of this section, TalentLMS presents its available tasks on a main section of the home page but also at a dropdown menu in the navigation bar option “Go to”. On the main page, independent of the role, a list of all tasks is presented organized by item, each section has a icon representing the item, a hyperlink with the name of the item and a list of hyperlinks with the tasks available for the item (Figure 2.5). This type of organization makes it possible for the user to know everything he/she can do in one place, without having an overwhelming feeling. On the dropdown menu only the icons and item names are shown.

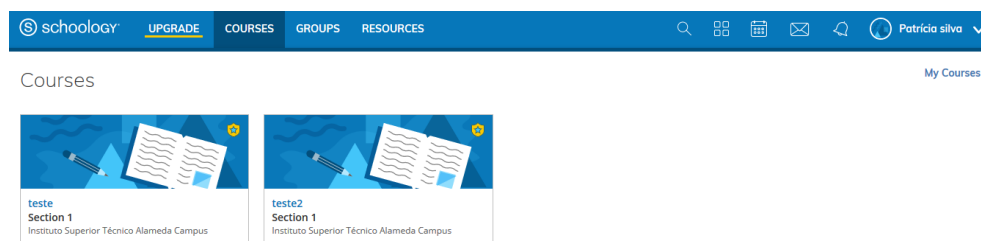


Figure 2.3: Schoology navigation bar and its second layer.

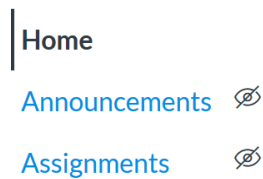


Figure 2.4: Menu inside Canvas course.

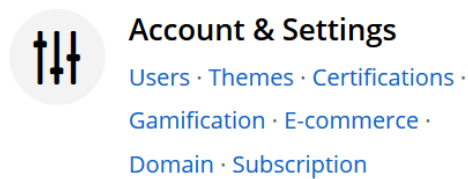


Figure 2.5: TalentLMS possible tasks.

The **menu path** is what helps the user know where he/she is within the system, it has the hierarchy of pages through which the user went to get to the current state. The most common and easy way to present it is using breadcrumbs (Figure 2.7), that we are used to see in the directories path of folders on our computers. The most complex systems in here use this to guide the user: Schoology, Moodle, Canvas, SapLitmos and TalentLMS. As simpler systems iSpring and Edmodo have no need to show this hierarchy of pages, Edmodo only shows the name of the course we are managing on the top of the page and iSpring while showing the name of the section we are in, provides us a "go back" arrow to come to the previous page. Blackboard is a more complex system and allows two types of courses: a older one that uses breadcrumbs as a menu path on the opening of new pages, and a more recent version that, for every new page open, uses a slider from the right and only a close button for it (Figure 2.6). When opening a lot of pages, all these sliders can be overwhelming.

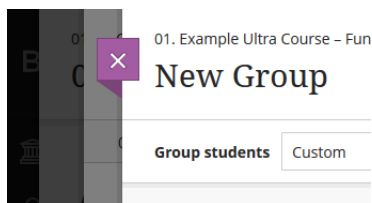


Figure 2.6: Layers of sliders on Blackboard.

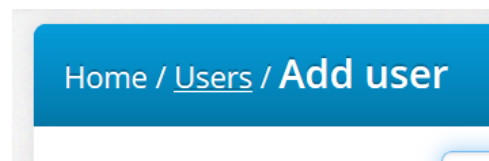


Figure 2.7: Menu path on TalentLMS.

2.2.2 Initialization

From the eight LMSs analysed in this section, SapLitmos, iSpring and TalentLMS are more focused on learning inside companies and firms than in schools, on their domain groups of users are assign to courses instead of having groups inside the courses. Thus, the word "**course**" in these systems means chapter of the course on education focused LMSs. On SapLitmos and iSpring there is no other item that resembles our definition of course, which means that per account there is only one course. TalentLMS offers the item "Branch", which is a way to have different environments with different students and course chapters. It is on the branch creation page where we can define a theme for the system (course) from a dropdown list, define user settings as the default user type and how they can be added to the branch, and select the gamification packaged all in dropdown lists.

To **create a course** in the remaining systems, we need to go to the correspondent listing page, which can be a table of the existing items (hyperlinks) and some definitions, a simple list of items in hyperlinks

or cards with some extra information on plain text, or a grid of cards with a preview image. Then we need to click on the add button: which is identified by either the standard icon "plus"; or plain text saying add or create course. Canvas uses the standard icon followed by the item name in all its add buttons to make sure the user adds the right item to the system. On Canvas, Edmodo and Schoology the add action will create a modal (Figure 2.9) with the basic course information of each system. These are not more than five inputs to make sure the action is quick and easy to accomplish.

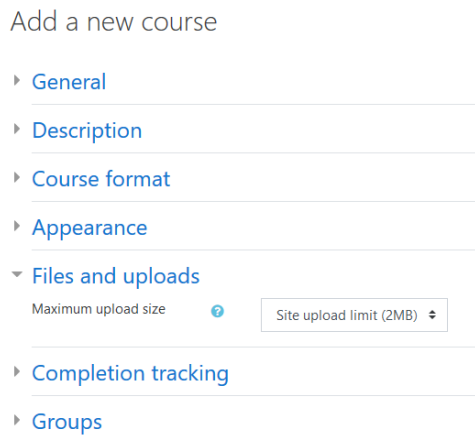


Figure 2.8: Create new course on Moodle.

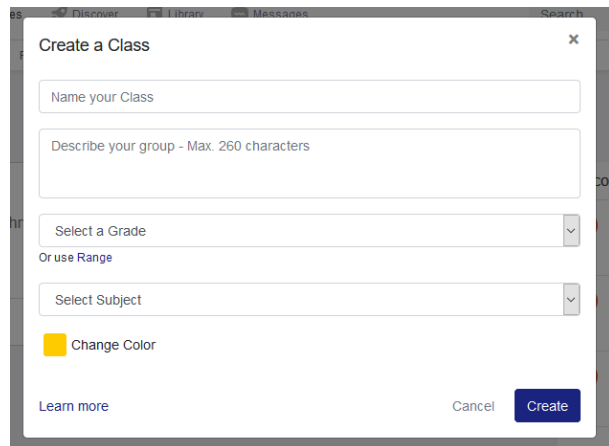


Figure 2.9: Create a new course on Edmodo.

On Moodle, the add button is under the listing instead of on the top of the page, making the user spend more time on this simple task. After clicking a new page is shown organized by sections displayed on an accordion (Figure 2.8). Even though there are a lot more inputs to fill than on the modals, the majority of them are dropdown lists which makes it a little less harsh on the user. This page is the same we see when selecting the edit settings option. The Blackboard tested on the platform mentioned does not allow new creation of courses, but comes with ten pre made examples that we can personalize to make our courses.

In LMSs, **user types and roles** are used to allow different views of the same thing. The way a Teacher sees a course is not the same as a Student or even a Guest. Roles are used to allow the hierarchical organization of users while at the same time define who can do and see what, since a student is not allowed to change the course content and the teacher is. User types are a specification of a role. They can be used to add more or less permissions to a certain role and, in some cases, change how a view is presented. All eight LMSs allow roles, having at least Teacher and Student. Edmodo and Schoology are the most basic case in which only these two roles (plus the Parent role for Edmodo) are allowed and as a Teacher, we can not change to the Student view and edit it. On the other six systems we can change between roles to see how the different views are presented.

On SapLitmos, Blackboard and Canvas the set of roles is fixed and we can only change between the roles of Teacher (or higher role) and Student. Moodle and iSpring let us add more roles, but also have the limitation of only allowing us to see the Student view besides our own. To change roles and see the

system as the user would see it, iSpring, SapLitmos and Moodle use a button on the dropdown menu from the profile icon (Figure 2.11), on the top right of the screen, and Canvas and Blackboard a button inside each course page (Figure 2.10). These last two systems are the only ones that when on Student view add a visual element that helps the user know in which role he/she is. Canvas adds a border to the whole screen in a vivid color and a bar at the bottom with two buttons, to either change back to original role or to reset the Student of the preview. Blackboard adds a bar at the top with the Student icon and a dropdown menu with the same options as Canvas, plus one to view the gradebook.

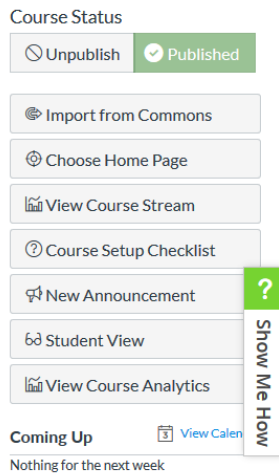


Figure 2.10: Change roles and views on Canvas.

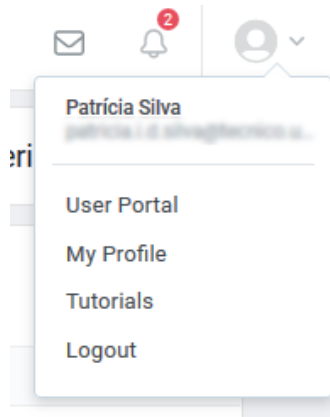


Figure 2.11: Change roles and views on iSpring.

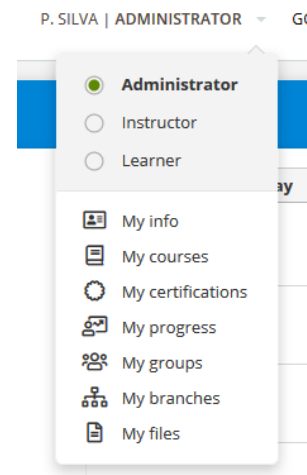


Figure 2.12: Change roles and views on TalentLMS.

To create a new role on iSpring, we need to give it a name and description (text field) and select which permissions do we want to give to that role. These permissions are organized by item and can be selected using a checkbox. On Moodle, we need to select the defaults of another role: by selecting from a dropdown list; or importing a preset file. Then we can specify the same information as in iSpring, say in which context the role can be assigned (checkboxes), which roles it can switch to, override and assign. On these last inputs (Figure 2.13) is not obvious what the user can do: while by clicking in different roles one at a time, only one gets selected (which resembles the radio button behavior); by clicking on one role and drag to another, both get selected; and using the control button pressed while selecting individually roles, all of them stay selected. The behavior resembles a checkbox list, which is actually being used outside the creation page, and should also be used here. The new role will show on the roles listing page. On iSpring we can access it through the tab roles on the users page. On Moodle, we have to first go to the "Site administration", to the users tab and then on the permissions section select "Define roles". Additionally, we can change the order of the roles in the hierarchy, change the ones that already existed on the system and delete some of them.

TalentLMS is the only one that offers the possibility to add user types and to change between all roles, below the original in the hierarchy. Even though there are only three roles, all of them have different views of the system. To change between them, we can use the radio button on the dropdown menu under the

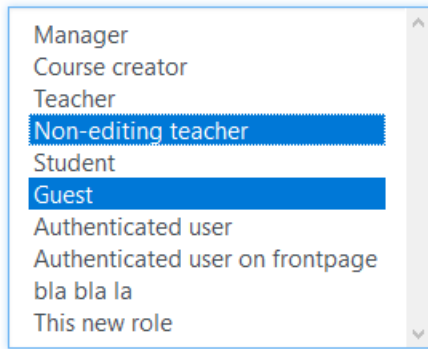


Figure 2.13: Moodle's role assignment.

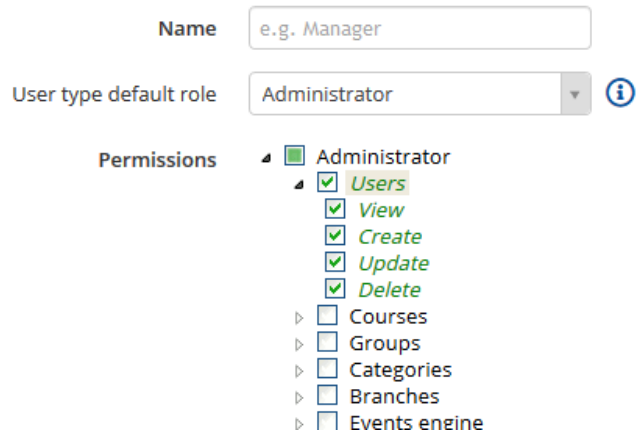


Figure 2.14: Creating a user type on TalentLMS.

profile name on the navigation bar (Figure 2.12). To create a new user type, we can select the task on the home page, or go to the user types listing page and there select the add new button. A flaw in the system can be detected if we try to access this listing page through the dropdown menu on the "Go to" option on the navigation bar, as user types is the only option from the home page that does not show here. While creating a new user type, besides giving it a name (text field) and selecting the base role through which the system will recognize the user, we have a collapsible checkbox list of permissions organized by role and item (Figure 2.14). When selecting a checkbox with child boxes, all children are selected. Thus, if we select a role, all its permissions are also selected. We can go very close in detail by opening every layer of collapsible checkboxes.

In each course page (education focus LMSs) or chapter of course (not education focus LMSs) we have access to a new menu with options to manage the course, personalize it, add content and in some cases this is also where we can **add users**. On Schoology, the only way to add a user is by sharing the access code, shown on a box in the course page. Users who try to then access the course by the code have to be accepted by the administrator (if that checkbox on the box is selected). Similar to it, Blackboard uses a link to provide access. This link is at the top of the users listing page and can be edited. On Canvas, when selecting the add button of the users a new modal shows, where we can add a single user or a group of users with the same role (dropdown list). Users identification can be selected on a radio button with the options: email address, login ID or SIS ID (Figure 2.16).

The other five LMSs allow us to manually add each user by filling in each text field (Figure 2.15) and selecting an option on some dropdown lists as the user type/role, in a new page. Edmodo besides manually input and code sharing, it also allows us to send emails of invitation to the course, by typing on email at a time on a text box. Moodle, iSpring and TalentLMS, provide an easier way to add users by importing them from a .csv or .xls file. On the case of the csv a dropdown list is presented to select the delimiter used. While every system does this add/import inside the course, Moodle does it on the "Site administration" section, users tab. Then on the course page a search on all users can be done to add the students to the course.

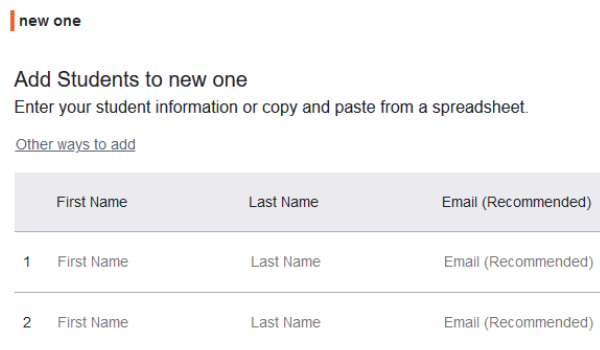


Figure 2.15: Add user one by one - Edmodo.

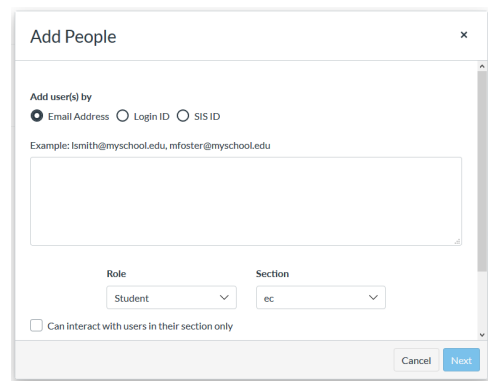


Figure 2.16: Add users by ID - Canvas.

A **profile page** is a page that represents a user and shows his/her information. Some systems choose to show more information than others, and that amount may also vary depending on who is seeing the page. Blackboard, Moodle and Canvas are the most simple systems in this case, showing only personal information and none about the course progress or achievements. When a user sees it own profile, a new page is shown with all the information the user can edit, organized by categories. On Blackboard and Moodle this information is shown in two columns and inside a table for each category, to edit a field we just need to select the pencil icon that shows up on hover (on Blackboard) or select the gear-wheel at the top of the page (on Moodle). This action will open: the same page where the user was created, on Moodle; and on Blackboard a slider at the right side of the page with all information on edit mode and the one, from which we selected the edit mode, on focus. On Canvas, as there is less information to edit, it is displayed in list format, and the edit button on the right corner of the page will transform the plain text in text fields.

When seeing other users profile page, Moodle acts in the same way as if it was our profile page. The other two systems both open a slider, at the right side of the screen, with the basic information of that user, Canvas shows some statistics and provides two buttons to explore those values, and Blackboard allows us, as Administrators, to change the user role. Schoology presents an equal page in both situations and very similar to the Canvas's self page. Besides the basic information it has a section for listing of the courses of the user and a tab menu to go to the updates section, where we can write anything, and the blog section, where we can create a post.

SapLitmos and iSpring behave like Canvas when we see our own profile page through the profile icon on the navigation bar, SapLitmos has an extra menu with jump links inside the page to make it easier to find the information and edit (Figure 2.17). If we select a user on the user list, even being ourselves, the profile page is shown in a different way. Both add a top bar with the name of the user and its role (SapLitmos also adds the email address), and below a tab menu to see all kind of information and statistics from personal information to achievements and recent activity (Figure 2.18). Edmodo profile page is always the same and its similar to the last page described. The tabs available change depending on whether the user is a teacher or student, an additional column with the connections of the user, using their profile picture, is shown next to the tabs section.

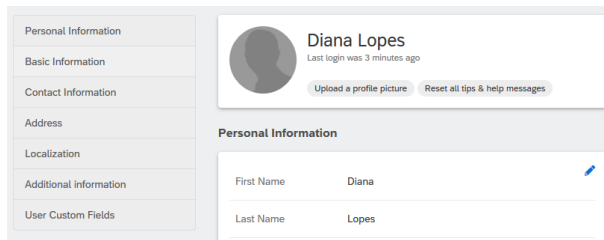


Figure 2.17: Profile page seen by self.

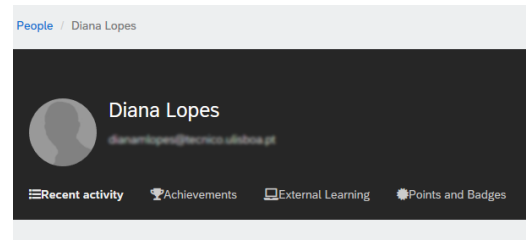


Figure 2.18: Profile seen through listing page.

On TalentLMS, the profile page is always the same but depending on who is seeing the profile some options can be visible or not. This page can be explored in two different menus: one tab menu at the left and a toggle menu at the right. The right menu selects the mode through which we see the profile and influences which options show at the left one (Figure 2.19). On Info-graphic mode, a more dynamic page with animations will be shown, with information about the progress of the user and no tabs will be displayed. On Progress mode three tabs are available, plus one if our top role is "Administrator", on these tabs the page shown is either a list of items or a grid of 2x2 with multiple type of visual information. An extra option on the right is shown if we are Administrators, so we can edit our profile information and see the lists of branches, courses and files.

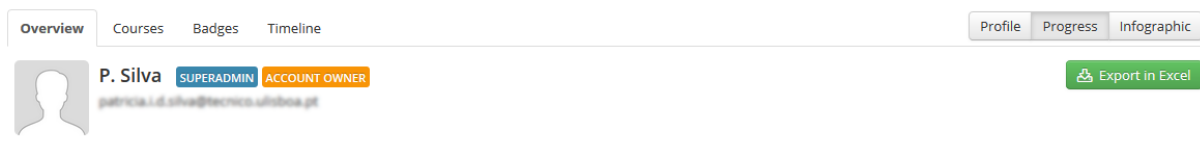


Figure 2.19: Top of the profile page on TalentLMS .

2.2.3 Customization

On these systems, we are allowed to do a little bit more than an online course and activate/deactivate some **extra features** to bring the course to the next level of engagement. All of them, besides Blackboard, support at least the feature Gamification with either points, badges, or both. Blackboard and Canvas have their feature configuration done inside the course settings section, and show what can be used in a list format (table for Blackboard and accordion for Canvas) with either a toggle button or a checkbox to activate the feature. Edmodo and Schoology use a special page of features that works like a shop, that we can access through the navigation bars. On both of them the options are listed using the name, short description and image preview. Before installing any new feature, a new page is open with full description and details about the app or pre made module. The last system with a specific page for extra features is SapLitmos, where each feature is displayed in a card with a icon, name description and a bar that says if it is active or not (Figure 2.20). By clicking on a card we go to a new page where there is at least a save button and a checkbox to activate/deactivate the feature.

Regarding **gamification**, Canvas is the system with fewer options. It only allows points that are given on assignments, through a text field on the assignment creation, which are used to calculate the

final grade on percentage. Edmodo uses points in the same way, but allows badges on specific goal points defined by the system, which can not be edited. On TalentLMS, besides badges and points, we can also use levels and leaderboards. To access and edit these elements, we need to go to the gamification tab on the account settings. On this page, settings are arranged by element, and each can be activated/deactivated with a toggle button or all at once with the main one at the top of the page. Each element can only be used in a set of cases presented below its section through a checkbox, but in some of them, the number of points given can be changed on a text field (Figure 2.21). In the badges, we can go to a customization page where we can change the name of the badge, change its icon by uploading a new one, or selecting a theme of badges from a dropdown list at the top of the page.

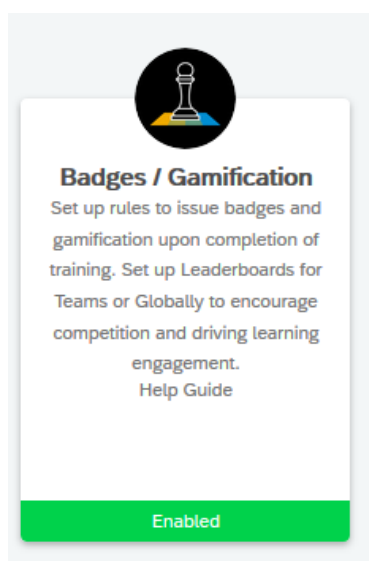


Figure 2.20: Extra features on SapLitmos.

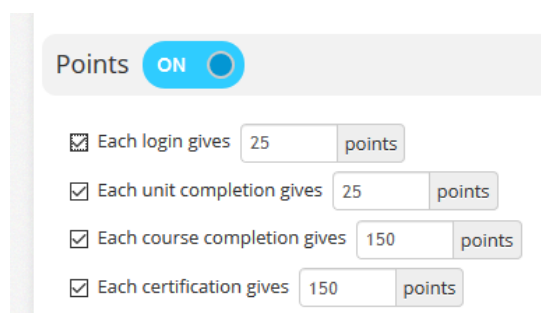


Figure 2.21: Points on TalentLMS.





	 Perfect Att...	 Good Liste...	 Homework	 Leadership
si, Pat	✔			

Figure 2.22: Badges assignment on Schoology.

On iSpring points work as on Edmodo and Canvas, given on assignments, but a default value can be set on the points tab of the gamification page of the settings. It is on the badges tab, that we can see our badges in a table and create a new one. For that a new page is open with text fields for name and description, a box with available images for badges and a button for upload, a dropdown list to select the rule that triggers the badge followed by either a text field if we select amount of points or a button if we select finish a chapter. SapLitmos has both points and badges that work as the badges from the iSpring system, with the extra of being able to change the color of the badge on the default list and the fact that points can also be given when triggering a badge.

Moodle only has badges and they can be fully customizable, from the look to when they are given, we can access the list of badges on the site administration right at the first tab or on the menu of a course. During creation all of the input are text field except from the button to upload a badge image and the expiry date of the badge that uses a picker. Finally, Schoology that let us manage points just like Canvas and create badges like iSpring. However here the badges can only be given manually through a

table where each column is a badge and row a student, by clicking on a cell we trigger the badge (Figure 2.22). On Schoology there are also some pre made badges that we can choose to use. By selecting "Schoology Badges" on the dropdown menu of the button "Add Badge", a modal is presented with a checkbox list of the available badges, already added badges are disabled and shown in grey.

As users of a system, **notifications** help us know what is happening and remind us of important occasions like assignments due date. SapLitmos and Blackboard don't appear to have notifications, but if they do we can not change them. On Canvas, Moodle, Schoology and Edmodo the notifications management is at the user level on the profile settings and there are a lot of options to explore in list format. To activate a notification Canvas uses a icon toggle button to choose the frequency to which the notification is received (a legend to the icons is presented on the top of the page - Figure 2.24). Edmodo uses a toggle on/off button and uses a tab divider to personalize the notifications for email and push messages. Moodle uses toggle buttons on a table divided by type of notification and when to receive it (Figure 2.23). Schoology uses a dropdown list with the options: on, off and for some cases custom which allow us to select the course where we want it active. In the end of the list there is a save button. The previous systems did not have a save button, saving automatically at every change.

	Web		Email	
Assignment	Online 	Offline 	Online 	Offline
Assignment notifications				

Figure 2.23: Notification on Moodle.

Notify me right away
 Send daily summary
 Send weekly summary
 Do not send me anything

Course Activities	Email Address <small>gabriel.l...@university.edu</small>
Due Date	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
Grading Policies	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
Course Content	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>

Figure 2.24: Notification on Canvas.

iSpring has a notifications settings page at each course, where we can change very little. However, there is one thing the others do not offer that iSpring does: the possibility to customize the email sent when a course is assigned to a user. TalentLMS treats notifications as items and lets us create notifications as we want. The listing page is a table with the notification name, the event that triggers it, and the recipient of the notification. When creating a new notification, we can choose the event and recipient from two dropdown list that, even though they are large, they are limited.

The last two things left to explore are **pages and themes**, with the goal of finding different forms to personalize our courses. From the eight systems, Edmodo is the only one that does not support pages,

and both Canvas, Blackboard and Schoology do not support themes. When creating a new page on TalentLMS, SapLitmos, Canvas, Schoology, Blackboard and Moodle, we have a text box with options to edit text like in Microsoft Word, and the option to change to HTML format. Even though we can add CSS on the HTML format, Blackboard also has a button to change to CSS format. Canvas has a shortcuts side menu at the right organized in tabs : files, links and images to easily add them to the page. iSpring offers a more dynamic way to create a page. It resembles to the Wordpress format, allowing cover images and using an add icon button, after the last paragraph, that shows a list menu with all sections available to add to the page. It includes images, citations, embedded code and others (Figure 2.25). The only thing missing here is the drag and drop feature of Wordpress, that allows us to better arrange each section within the page.

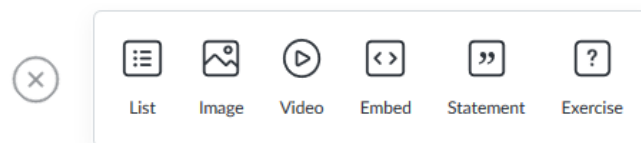


Figure 2.25: Add content of a page on iSpring.

Besides pages, BlackBoard on the older and more complex version of a course also let us change the organization of some elements. We can drag and drop the elements on the course menu to change their order and click on them to change their name (a possible way to correct the flaw shown on Thacker's study [24]). On the main page of the course, we can also drag and drop elements and hide them.

Regarding themes, iSpring and Edmodo only allow us to change the course's color, picking from a small set of colors. Moodle lets us change between pre-installed themes by showing them on a list with a preview on the side and a button to choose it. If we allow it on the settings, we can also provide a theme file specifically for a course on its creation page. On TalentLMS, we can change our branch's theme by selecting one on a dropdown list while editing. This list is not closed as we can go to themes on "Account settings" and personalize an existing theme by changing its colors or writing a specific CSS or Javascript. Those changes can be saved in a new theme to use later.

In the first section about home pages, we explored the option to change the way we see that page on the administrator side of SapLitmos. However, we can also change some aspects of the course's theme and some parts of the learner's home page (Figure 2.26). On the account settings, we have a section called themes, where we can select the layout of the course/system on a dropdown list and change its colors on a text field. From a large number of checkboxes, we can choose what is visible on the student's dashboard and even add a custom banner, header, and footer by writing HTML on each text box and CSS on another one. All these changes can be seen in a preview on the right side of the page.

2.3 Discussion

The studies of the section 2.1 helped identify the main problems of usability and UX on four of the chosen LMSs (Table 2.6). We knew since the first article that BlackBoard was not the best LMS, but it was with

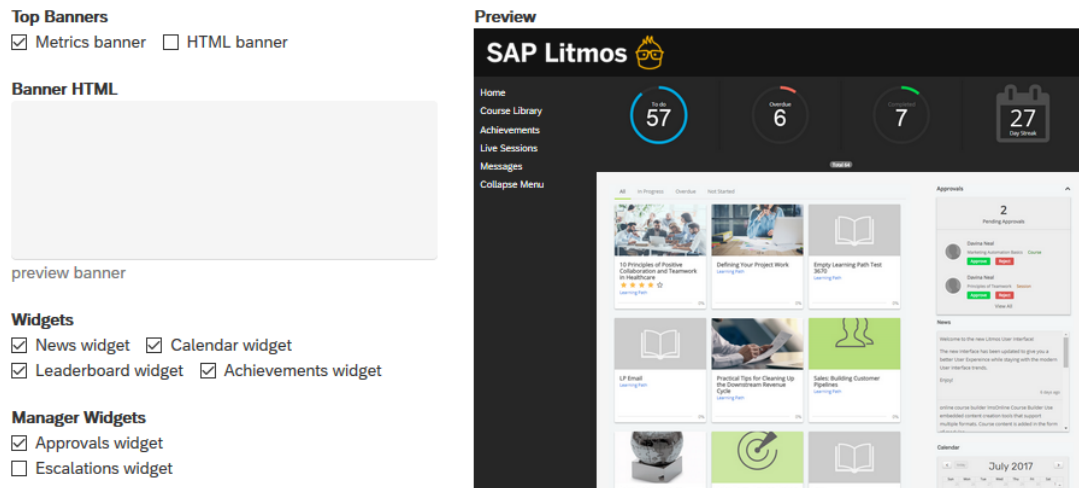


Figure 2.26: Theme customization on SapLitmos.

a later study that we really saw the main flaws of this system in terms of usability. Eventhough Moodle is one of the most used LMS, it is not perfect and there are still some points in which it could be improved. Canvas and Schoology are with no doubt the systems that stand out on this studies, and that may be because they are the newest of these four (Canvas - 2012 and Schoology - 2014).

LMS	Usability Problems	UX Problems
Moodle	H8 and H4	Novelty
BlackBoard	H2, H3, H5 and H9	-
Canvas	H4	-
Schoology	-	Novelty

Table 2.6: Final results from the four studies.

Through the visual analyse of these systems, we were able to see some changes to correct the flaws mentioned: Canvas replaced it's grey hyperlinks by a standards icon for not visible, the crossed eye; and Blackboard allow us to change the names shown on the menu to make them more clear and resemble the real world scenario. This analysed was also important to identify patterns between systems, GUI elements that may be useful for the Gamecourse project.

Table 2.7 is a summary of the structure of these eight systems. It is clear that breadcrumbs are the best way to present the menu path on a complex system. TalentLMS approach on showing the available tasks is not common but works very well as a home page, when items are not restrict to one particular course, specially on the Administrator/Teacher side. For the Student side a list of courses using cards can be an easy way to select the desired course. The second layer on the menu navigation bar is also an interesting element, not only to display a list of items but also for the list of tasks available for each item. However these lists have to be of small or average size to make the interaction simple and fast.

In terms of inputs all the systems work with the defaults: text fields for free input; checkboxes for true or false purposes or checking multiple items of a list; dropdown list to select one of multiple options, usually on big lists; radio button also to select one of multiple options but on smaller lists; and toggle

LMS	Home Page	Possible Tasks	Menu Path
TalentLMS	Student - list of courses + statistics; Admin - tasks + statistics	Dropdown menu + list on home page	Breadcrumb
SapLitmos	Student - list of courses; Admin - custom	Top navbar	Breadcrumb
Moodle	List of courses + custom blocks	Side navbar	Breadcrumb
Canvas	List of courses	Side navbar + highlight + second layer	Breadcrumb
BlackBoard	List of courses	Side navbar + highlight	Breadcrumb or sliders
Schoology	News feed or list of courses + calendar	Top navbar + second layer	Breadcrumb
Edmodo	News feed	Top navbar	-
iSpring	Student - list of courses ; Teacher - statistics + content	Side navbar with highlight	-

Table 2.7: Structure analyse of the eight LMSs.

buttons to on/off scenarios. The listing pages are also simple using either tables, cards or accordions. Even though tables are older elements, when there is a need to compare details of the items they are the best option.

On the creation of an item two possibilities are presented: a modal and a new page (Table 2.8). Modals work better when the amount of inputs is low, below five as we saw, and new pages for a large amount of inputs and configurations. In the majority of the cases the creation and the edit page/modal is the same, but these two elements could work together to first gather the essential information (on a modal) and then allow more details and configuration (new page). Although sliders also work very well when presenting new information, like Canvas does to show other users. When used too much they can create visual noise, like Blackboard that allows four layers or more to be visible at once.

When referring to users, using a collapsible checkbox list for permissions looks like a good option, if organized by item. Changing role in the GameCourse project would be more useful during the preview of custom pages and views, together with a top bar saying each type of user or role is being used to see the page. Color border would only work with one possible change between roles and user types, otherwise would make the user have to remember the colors.

Extra features and modules could be presented in a usual listing page, but SapLitmos approach does it better by using cards with icons and status bar (Table 2.9). The status bar is a little detail that gives immediate information to user with no need for extra clicks. Notifications can be done either at user or course level, each has its advantages, but in terms of GUI elements they can both be designed in the same way: a list or table with toggle buttons depending if there is more than one type of notification.

Pages are in the majority of the cases just personalized with text, html and css, creating only static and limited pages. None of these systems offer a dynamic page, but SapLitmos lets us personalize both theme and administrator main page with more options. The drag and drop system works well with limited

LMS	New Course	Roles & User types	Change Role	Add Users	Profile
TalentLMS	Branch - 5+ inputs (N)	base role + checkbox list	Radio button (P)	Manual or file (N)	Toggle menu + tabs
SapLitmos	-	-	Button (P)	Manual (N)	Self - PI (list + jump link); Other - top bar + tabs
Moodle	10+ inputs, accordion (N)	base role + checkbox list	Button (P) + list of buttons	Manual or file (N)	PI (table)
Canvas	3-5 inputs (M)	-	Button (C) + border	Manual (M)	Self - PI (list); Other - slider
BlackBoard	-	-	Button (C) + top bar	Editable link	Self - PI (table); Other or edit - slider
Schoology	3-5 inputs (M)	-	-	Access code	PI (table) + course list + announcements
Edmodo	3-5 inputs (M)	-	-	Manual or file or code (M)	Top bar + tabs + connections box
iSpring	-	Checkbox list	Button (P)	Manual or file (N)	Self - PI (list); Other - top bar + tabs

Table 2.8: Initialization analyse of the eight LMSs.

P - dropdown menu on profile icon; C - inside a course; N - new page;
M - modal; PI - personal information.

option of blocks and the preview makes it easier to the user to know what the final page will look like. In this case, modals and sliders could also bring some extra functionality allowing to edit details on blocks without having to save and leave the page.

Moodle lets us do even more than what was explored. However, this ability to make anything, leaves the system too complex to understand and work with on the Administrator level. One thing to point out about Moodle is the excess of buttons on the system: if we want to add new content to the course, we first need to click on the edit mode button, which then adds a button inside the course to add content. An unnecessary extra click that, without exploring, would not even be clear.

LMS	Extra feature	Gamification	Notifications	Pages	Theme
TalentLMS	-	Toggle button + checkbox list; Badge C2	Create item + dropdown list (C)	Basic	Existing themes + css + js + colors (Hex)
SapLitmos	Cards + status bar	Points; Badge C1 + limited triggers	-	Basic + custom home page	Html + css + colors (Hex) + checkboxes + preview
Moodle	-	Badge C3	Table + toggle button (U)	Basic	List + preview + import
Canvas	Accordion + toggle button	Points	Table + toggle menu (U)	Basic + shortcuts menu	-
BlackBoard	table + checkboxes	-	-	Basic + css & drag and drop	-
Schoology	Shop format	Points; Badge D + C2 + manual trigger	List + dropdown + modal (U)	Basic	-
Edmodo	Shop format	Points; Badge D	List + toggle button (U)	-	List of colors
iSpring	-	Point; Badge C3	Custom email (C)	Wordpress format	List of colors

Table 2.9: Customization analyse of the eight LMSs.

C - course level; U - user level; Hex - hexadecimal format;
 Badge D - default badges available; Badge C1 - change icon;
 Badge C2 - C1 + change name of badge; Badge C3 - C2 + change trigger event;

Chapter 3

The GameCourse system

GameCourse is a Learning Management system with gamification aspects that allows us to create and manage our courses. Its goal is to present a course experience to students that feels like a game that is also personalized depending on the learning style of the student. This system has been used for ten years on the Multimedia Content Production course from the MSc. of Information Systems and Computer Engineering. Many features and adjustments have been implemented throughout the last years. However, the UI has only had two versions, and it has been the same since 2016, the year André Baltazar developed Smartboards and improved the system both visually and functionally [12].

3.1 How it works

On the GameCourse system we had several concepts that defined the system's structure. The *Course*, one of the main elements of the system, where all the base information of a course was saved and used as point of reference to add additional information and functionality. Another main element was the *User*, which kept the base information of a user and defined if it had admin privileges or not. Then we had *Course User*, the connection between the last two concepts, that defined the user within a course saving all the information that is exclusive to a course. Then we had *Roles* that can be attributed to a *Course User* to organize the users within groups. There were three main *Roles* the Teacher, Student, and Watcher, the first one gives the user permissions to configure the course, but more could be created. We also had *Modules* that could be activated within a course and extend its functionalities.

One of the existing *Modules* was called *Views*, which allowed the creation of HTML structure that could be used on *Templates* or *Pages*. Associated with *Views* we had *Aspects* that created a variation of the view depending on who was seeing it and who it belonged to. Each view was built with the help of four different *Parts*: Text; Image; Block; and Table, that created the base structure. On those *Parts* it could be used the *Expression Language*, which fetched information from the system to be displayed or iterated, to create multiple elements with the same structure. While *Views* could be used on *Pages* to created another course page, it also could be used on *Templates* that could be reused within the course or the system, if globalized.

On the server side all these concepts and correspondent information were arranged and managed trough a MySQL database (Appendix I), PHP classes (version 5.6), and external scripts that would fetch information from external data sources, like user's information and module's specifications. On the client side these concepts were shown and manipulated through the correspondent listing and action pages managed with AngularJS 1, which created and rendered the GUI. To enable communication between both sides there was an Application Program Interface (API), where the available functions and their functionality were registered. As well as an app service, from which the client side could make a request

to the target API function to obtain its provided data. The main focus of this project is on the client side and on the API definition as new information and functionalities are required.

3.2 Current UI

The initial UI was simple and minimalistic. It used a monochromatic color scheme with no distinction between sections or courses: the system looked the same whether we were inside a course settings page (Figure 3.3) or inside the system setting page (Figure 3.4). The menu was shown on a top navigation bar with highlight for the selected section and, depending on whether we were inside or out of a course, the options on this menu change. Inside the settings page there was also a side menu with the components of the course or system. These components were organized by type and shown with indentation to visually organize every item (Figure 3.1).

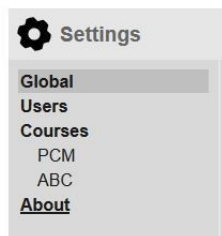


Figure 3.1: Side menu - system domain.

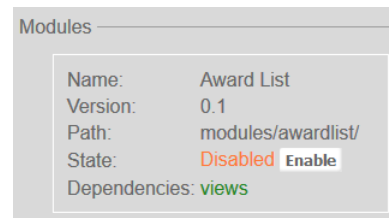


Figure 3.2: Modules on the settings page.



Figure 3.3: Navigation bar - system domain.



Figure 3.4: Navigation bar - course domain.

All the UI elements were presented inside boxes making the system look too geometric/squared, especially on the view builder where there can be many elements (Figure 3.5). New actions on the system triggered a new page, changed a part of a page, or added an overlay. Both themes and modules were displayed using cards (Figure 3.2), but the other listing pages were a simple list of text elements. To change the list of users inside each course we had to do it by role on a single text box.

The edit view page was organized by elements and actions (Figure 3.5). Depending on the type of element we could add some new elements inside or define the content to be shown on that area. The actions of each element could be selected from a toolbar while hovering on the element. From the set of actions we had: the "edit layout" button, that allowed us to add new elements and drag and drop them to rearrange their order; the "save template" button to save the element and its children as a template; and the settings button which allowed us to define the content and action of the element, such as the data through which we wanted to loop, creating a list of similar elements, and the specific content we wanted to display on the element.

To determine which data is displayed on each element we had to use the expression language, implemented by Alice Dourado on the last version of this system [11]. To know which variables and

functions we could use we had to select the question mark button which would lead us to a extensive documentation page with information about the language and how to use it (Figure 3.8). On each field of the settings page we had a status icon that told us if the syntax of the expression was well written or not, by using the color green and red (Figure 3.6 and 3.7).

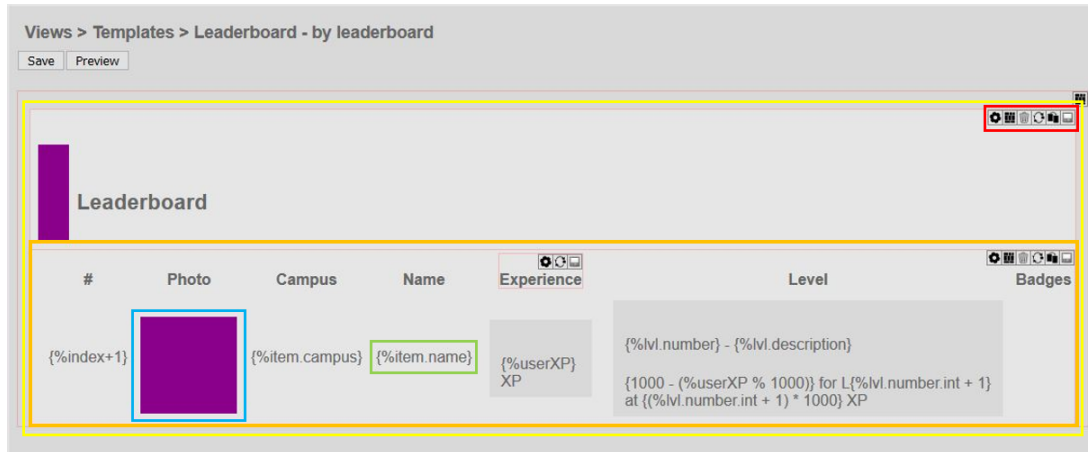


Figure 3.5: View editor with table (orange), block (yellow), text (green), image (blue) and edit buttons (red).

During the tests made by Alice to validate her work, some questions and concerns about the UI of the edit view page [11]. Users complained about the buttons on each element toolbar, saying they were too small and not clear to which action they represented. They had trouble finding out that to edit an element we needed to get out of the edit mode so we could see the toolbar and select the action wanted. While on the setting page of an element, users were confuse by the functionality of the button close, which also performed the action save, having no way of canceling what was done. Some concerns were also presented regarding the feedback of the system. Even though there was an icon on every entry of the expression language to tell if it was wrong or right, when the expression was wrong there was nothing to help the user correct it. When that field is a variable there was also nothing to tell if the variable had been defined. Alice's work was helpful to discover current problems on the UI of the system, but as her tests only covered the View editor page, we still needed to perform a user tests through the whole system to collect the rest of the problems and Requirements of this project.

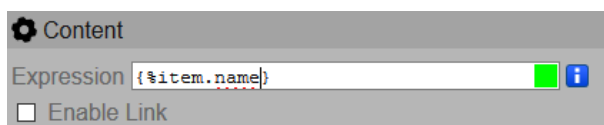


Figure 3.6: No syntax error on expression.



Figure 3.7: Syntax error on expression.

[Docs](#) | [Functions](#)

Views

Views allow control over what is displayed in the system's

Aspects

The Aspects are specializations of the views for specific versions of a page that can be displayed according to the page.

Figure 3.8: Documentation tabs.

3.3 Preliminary work with User Tests

To collect more information about the UI problems of the current system, we conducted User tests with a group of ten users. From these ten users, eight of them are current computer science students, one is a teacher and the last one has no connection with the computer science field. On the tests we used the think-a-loud, direct observation and interview methods to collect as much information as possible from the user experience with the system. To make sure the information collected was accurate, the tests were performed in an isolated room with no distractions and the voice of the user and his/her interaction with the system were recorded. For that, we used a voice recording app on a smartphone and the Xbox screen recording program on a computer, alongside with the current version of the system.

Before each test we read a quick introduction to the user to inform him/her of the objectives of the test and ask for permission to record the interaction with the system together with his/her voice (Appendix C.1). Each test was composed by six tasks, with various levels of complexity, randomly delivered to the user during the test (Appendix C.2): Listing the current courses (T1); Activate a module (T2); Create a new course (T3); Add a new student (T4); Change role of a student (T5); and Create a new page and correspondent view (T6). The final part of the tests included an interview regarding four categories of analysis, each one of them with three or four questions (Appendix C.3).

3.3.1 Results

T1 - Listing the current Courses: This task was the most simple one, only requiring the user to select "Courses" on the navigation bar. Therefore there were no apparent problems and every user managed to complete it successfully. Later on the interview when asked about the look of the system one user mentioned "it would be nice to have more information about each course and somehow highlight them".

T2 - Activate a module: On T2 users needed to explore more to get to the right page, they had to first select the course intended from the previous list, then go to the settings and there find the right module to activate (Figure 3.2). All users succeeded on the task, but only 40% completed the task with no problem, 30% were confused while searching for the right module to activate and 30% had troubles finding the right settings page. One of them mentioned "it was not obvious to me that the same button - settings - would lead to different pages".

T3 - Create new course: While on T2 the user needed to first select a course to then select the settings, on T3 the user needs to select the system settings instead. There he/she will need to select the courses tab where an "Create New" will be available. Even though 100% users completed this task, all of them tried to do it on the courses listing page, searching for an Add button there, and one of them got lost between the setting pages.

T4 - Add a new student: On T4, users had to go to the settings page of the intended course and select the option "Students" under "Configurations". There he/she had to add a line on the text box with the information we provided about the new user, respecting the format displayed. Then click on the

button "Replace all" to update the list of students (Figure 3.9). All these little actions made T4 present more problems than the previous tasks. Although 100% of the users completed the task: 30% had trouble finding the correct page; 100% got confused with the button "Replace all", as they were looking for an "Add" button instead; and 50% showed frustration towards the way of inserting data. One user said, "even with the format, that was not very visible, I would not have been able to fill the information if there were no previous examples there" and other suggested that "Add users should be outside of the settings pages, is not intuitive, and instead of one field it would be better to have several fields to fill in". While doing it, 30% users got an error message, one being user friendly (Figure 3.13) and the other being an error from the DataBase that only a developer from the system could understand. These same users later, on the interview, reported that the system needs more helpful feedback, "I only understood the error because I am a programmer and it was not even easy for me to read it" (Figure 3.14).

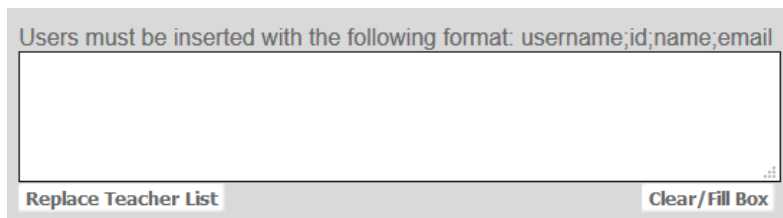


Figure 3.9: Text box to insert new user (Teacher).

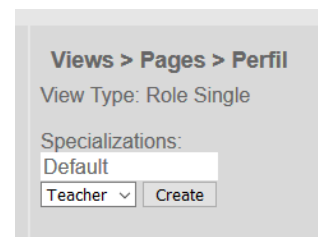


Figure 3.10: Middle page to edit page/view.

T5 - Change a role of a student: To change a role of any user, first the user needs to select the target course and the correspondent setting page, then click on the tab "Roles" to access the user list and change their roles (Figure 3.11). The major problem on this task was finding the right page to complete the task, having 100% tried to access the sub pages of the "Roles" section on the side menu and 80% tried to do it through the listing pages of Students and Teachers. All of them reported that it was not clear that "Roles" was clickable, "when I clicked on - Configurations - nothing changed so I assumed only the under tabs were clickable". Besides that, 50% users got confused with the hierarchy section of roles (Figure 3.12), having three of them tried to save progress through it, "I was expecting a save button at the end of the page and not in the middle of it". Users also showed some confusion with the feedback on this page, "I was expecting a pop up to show just like in the other pages to know if it really changed what I did".

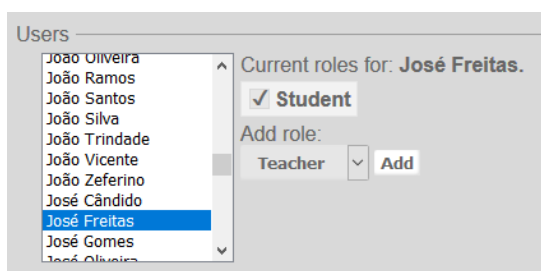


Figure 3.11: Change user roles.

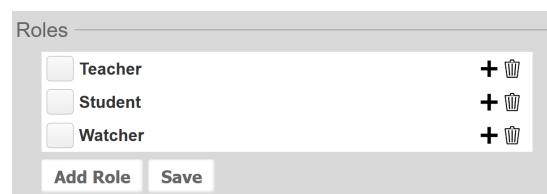


Figure 3.12: Roles hierarchy.

T6 - Create a new page and correspondent view: On T6 all the users succeeded on creating a new page, however only 30% managed to fully complete the editing of the page. To edit a page the user needs to first go through another page, where the "Specifications" of the view are defined and he/she has to click on "Default" field to move forward (Figure 3.10). This is a confusing step and 100% users thought they were on the wrong page, 30% even tried to access it through the navigation bar, "The Default field looks like a text box not a button", "I clicked in it by accident". Besides that, 80% users got confused and "afraid", when finally entering the edit page, when seeing the "No Children" label on red, making them go back, "it looked like an error, it felt like I did something wrong so I went back".

After selecting the "Default" field users had to click on the "edit layout" button to be able to add an image and text field to the view (Figure 3.5), then they had to click again on the "edit layout" button to be able to access the configuration buttons of the elements. 80% users reported that the buttons were too small and difficult to access "I had to be careful when moving the mouse so the options wouldn't disappear". While on an element configuration page, 40% were not sure if the close button would save the progress or not, "I had to come back to make sure the information was saved".

Regarding the use of the expression language 90% had problems with it when selecting the content of an element, "we should be able to have some kind of preview of the result of the expression", "it would be helpful to have more feedback or to have something like an auto-complete". To try to get some help, 80% users went to the documentation, from which six had trouble navigation through it (Figure 3.8), "I did not notice the tabs in the top part of the page", "the information is boring, it does not help when you don't know what to do, there should be some kind of flow or video explaining the basics". One user said the expression language "is too hard to understand, the learning curve is slow" and suggested that instead we should use a visual interface to select the content of an element.

During the interview we asked the users what was their initial and final thought about the look of the system and none of them liked it: "Looks like an old system", "Looks like someone forgot to update it", "It is simple but not in a good way". Some of them explained why they waited so much time on each page before going to another "I was not sure if the page was fully loaded or not, it looked unfinished".

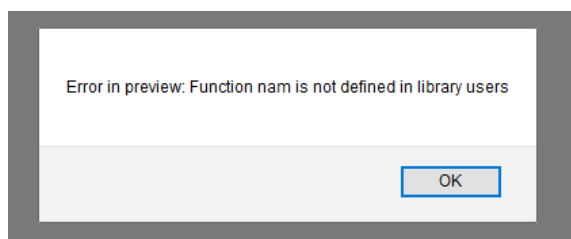


Figure 3.13: Error that helps the user understand his/her error.

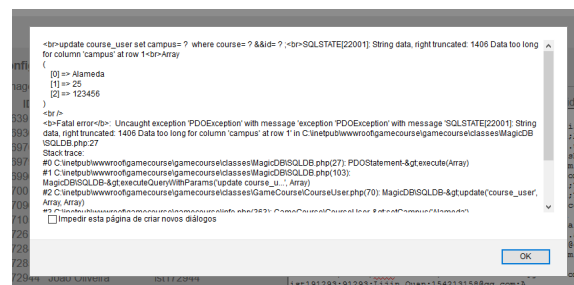


Figure 3.14: Error with information only a programmer can understand.

3.3.2 Requirements

With all the information collected during the user tests and with the feedback given on Alice's project, we were able to identify the main problems of the initial UI of the system. Besides a new visual look, so it stops looking "old" and "outdated" as users reported, the UI will need a lot of changes. We gather them in the project requirements described below.

Pages organization: It was clear that the current page organization is not clear and creates some doubts and confusion between the users. To fix that a new organization will be arranged to put together all the related information, divide non related information and make sure the flow of the system is correct. New subsections will also be created to complement and complete this requirement.

Setting page redesign: As the organization of the pages suffers from changes, so will the setting page, its menu and the sub-pages within (including modules configuration). A new design will be implemented so the menu itself is more clear and familiar, labels self explanatory and all the information is easy to see and to access.

Documentation page: The documentation page is an important page to beginners, but it must be useful and simple to use. For that reason a new documentation page will be built to summarize the information on subject and sections of the editing element page, it will have a fixed menu to be more visible than the current one and each section must not occupy more than what a user can see at once.

View editing page: As the most complex and confusing page it needs a lot of fixes. The way an element is selected will be changed so it is obvious from the beginning that elements, and which ones, can be added to the view. The access to this page will be redesign so it is more obvious to the user that he/she can visually change the it. To increase the level of help in the edit element page, the auto-complete will be added to the expression language fields, a slider will be introduced with the list of functions and variables available on the system, for quick access. To finish, a visual tutorial explaining each field of the page will be implemented to quickly introduce these two pages to a new user.

Modules configuration pages: To remove the external script that gathers information about the modules, configuration pages will be added to the modules. This will be done with a modular system that will generate a configuration for each module that requires it, including the existing modules and future ones.

Changes related to new features: At the same time that we will be creating a new UI for the Game-Course system, some new features will also be implemented. Among them: the ability to import/export elements of the system; the creation of plugin layers for data import and authentication; and the creation of teams. These and other new features will need new GUI elements to make them available to the user. Depending on changes of implementation of these features the design might change more than once during the development.

Chapter 4

GUI Design

In this chapter, we will explain the first phase of the project, which includes the design process of the new GUI of the GameCourse system, the Low fidelity Prototype (LFP), and the Heuristic evaluation with UI & UX experts.

4.1 Design Process

Before styling any part of the system we had to create the new page organization. It was clear during the User tests, performed on the initial GUI of the system, that not all actions related are near each other. For example, if you wanted to visit a course you would have to go to the "Courses" page and then select the desired course, whereas if you want to create a new course you would have to go to the "settings" page, select the "courses" tab and then select the add new button. The same happens to users and their roles inside a course. To fix this, on the new GUI, each of the pages of the system has its own actions and domain associated (Table 4.1): every action related to courses is available on the "Courses" page, every action related to users is on the "Users" pages, etc.

Scope	Pages	Actions
System	Main Page	
System	Courses	add, duplicate, edit, delete, access, import, export
System	Users	add, edit, delete, give/remove admin permissions, import, export
System Settings	Installed Modules	import, export
Course	Users	add (new and existing), edit, remove, add/remove roles, import, export
Course Settings	Roles	create, move, delete, define landing page
Course Settings	Modules	enable/disable, configure
Course Settings	Views	edit, create page/template, create/change specialization, globalize template, use global template

Table 4.1: Page organization on the new GUI.

The first drafts of the GUI were done on paper, so we could quickly and without much precision put out all of our ideas about the system's different and principal components. Then we moved into an online design workspace called Whimsical¹, which allows us to build various schemes, including wireframes, where we created the pages' structure while having a real notion of sizes and proportions of elements and white space. Even though this program has some limitations and is not as complete as Sketch (one of the most used design tools on the market), it makes it easier for developers who only have basic knowledge of design to build a coherent GUI without a professional designer. Elements can be designed from a big list of available shapes, pre-made components, and icons, making the design process less stressful and faster. It is easier to select one possibility within 100 than within 10000 possibilities.

¹Last version of the wireframes on whimsical: <https://whimsical.com/tese-Tv7J6qNoKvSuyuPwY5ZME9>

4.1.1 Modals

The main goal was to make the GUI look simple and minimalist, but at the same time have interesting elements to keep the user enthusiastic about using the system. Having that in mind and inspired by the analyzed LMSs, we decided that actions that require input from the user would be handled through a modal dialogue box where all the necessary inputs are available to fill, instead of opening a new page. In this way the user would still have context from the page where he/she selected the action.

Before styling the modals, we had to decide how inputs would look. Between the possibility of having a label associated with the input or just a placeholder, and being a full outlined box or just with an underline, we decided to use only the placeholder and use the whole input outlined to visually match an image input that would be represented by its preview in a box. We started by using a checkbox for the boolean type of information but later gave them the same design used in the display tables. When choosing the modal layout, we first designed it with a header, with a bottom line and text that would almost meet the element's margin. However, it was changed to a more clean design where the caption is shorter and aligned with the input fields, leaving an equal margin along the whole modal (Figure 4.1). We also added a close button on the top right corner and a "save" button on the bottom right corner, as the conventions do. To finish the general look of modals, we added a semi-transparent background layer to keep the focus on the modal information but still have context from the whole page.

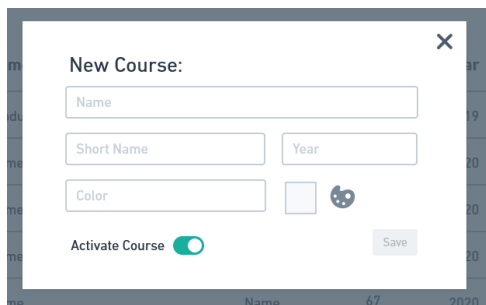


Figure 4.1: Add new modal.

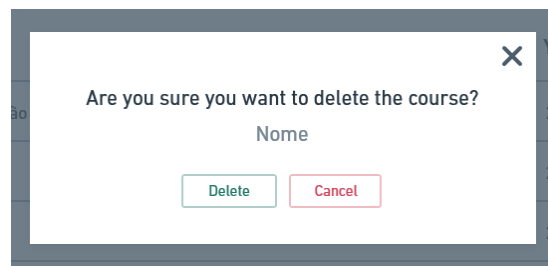


Figure 4.2: Verification modal.

To prevent errors, we also created a verification modal so the user can confirm his/her destructive action before it is too late. Contrary to the previous modal, we placed all the information centered and gave the color red to the "cancel" button and the color green to the "delete" one (Figure 4.2). This way, there is no implied information due to the location of the buttons.

4.1.2 Menus

From the initial GUI, we decided to keep some of the visual structure: the navbar on the top of the page, as it is the most common place for it to be; and the menu inside the settings as a sidebar, making it easier to navigate between pages within the settings. However, the style of these two elements was not kept. Regarding the navbar, we explored the following options:

- Each page identified only by an icon, which could work on the system domain. However, inside

a course, as the admin users have the freedom to create pages, the icon choice would be theirs, and that could create confusion, as it might not be coherent with what we are designing now;

- Each page identified by icon and text when hovered or selected. Although it would make it easier to associate the page with its icon and take advantage of recognition rather than recall principal, it could create missing clicks when the sections would open and close due to the hover effect;
- Each page identified by text. Nothing moves out of place, and we can still use style to distinguish selected pages from non-selected ones.

We chose the last option and used the page title with terms that are already familiar to the user and resemble the real world. Instead of placing every option on the right side of the navbar, we centered all options, separated them with a dot, and added extra elements to the side: the GameCourse new logo to the left and log-in information to the right with a logout button.

Regarding the sidebar, we were divided between using a tab style that recalls physical folder tabs or using white space as a divider and underlining the title of the selected section. To keep it on the minimalistic side, we went for the second option and later, for consistency, applied the same concept on the navbar (Figure 4.3) instead of using dots.



Figure 4.3: Navbar using white space as dividers.

4.1.3 Listing pages

Both the "Users" and "Courses" pages are intended for object manipulation, in which the quantity of objects can scale very fast. For that reason we decided to go with a table to display all the information needed at once and allow the needed actions per object. This table could look like an excel table and have a border on every single cell, but instead we went for the look that Canvas has, where the border is only placed per object, horizontally, separating it from the previous one. It gives a clean look while easily informing which information belongs to which object (Figure 4.5). To keep the distinction between actions and information display (here and in the rest of the system) instead of using buttons with text we added standard icons, that follow platform conventions, for each action (Figure 4.4): add: plus icon; edit: pencil icon; delete/remove: trash icon; export: arrow going out of a box; import: arrow going into a box; and duplicate: double boxes;

Inside a course the "Users" page has an extra information regarding the role of the user. To display it on the table we used tags after the name of the user, just like in TalentLMS. On the edit and add modal we added a section to change this information, similar to the one that existed on the roles page of the initial GUI: a list of clickable tags to remove the role and a dropdown to select a role to add.

Another important aspect of the design of these tables was Boolean object attributes, that translate to true or false options. The questions was if we would use plain text as the rest of the information



Figure 4.4: Standard icons used.

(like Canvas), or a check icon and make it slightly different from the action buttons (like Schoology), or perhaps incorporate an action and display at the same time using a checkbox. But none of them seemed interesting enough. Thinking of the IOS system we realised that, with an on/off button it was possible to have a visually interesting element, that would transmit the right information (checked and unchecked) and allow action at the same time. That was our chosen option as it maximizes the efficiency and flexibility of changing this values (Figure 4.5). No need to edit the object and save it, two less clicks.

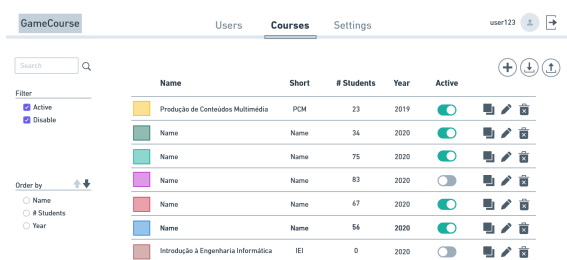


Figure 4.5: Courses page for admin users.

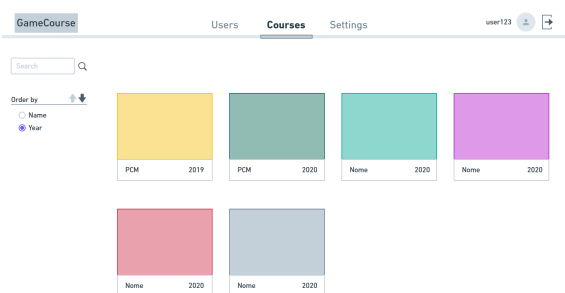


Figure 4.6: Courses page for non admin users.

To make it easy to find a specific object on the lists, we added an option to search, filter, and order the objects' list. Between a collapsible sidebar to the left side, a dropdown menu coming from the top left, and a fixed sidebar, we went for the last option. A fixed sidebar allows quick access to the options without stealing the attention from the page's main focus, the list, and avoids the overwhelming feeling of too many things changing on the same page (Figure 4.5).

All these pages are designed thinking of the admin user and his/her permissions. However, the "Courses" page is also visible to non-admin users. Having that in mind, we created a non-admin version of the page where each object is represented in a card showing the course color in evidence and below its short name and year (Figure 4.6).

On the "Modules" and "Views" the number of objects does not scale so quickly (they require more attention and time to be managed), have visual elements that identify them, and have less information. So, we decided to use cards instead of tables, with the object's visual identity on top and information and/or actions below. Like the cards used on SapLitmos, on the modules (Figure 4.8), and the cards we used for the non-admin version of the "Courses" page, on the views.

On all these pages we placed the action buttons on the top right of the page, keeping some distance from the navbar to have a clear view and to visually inform the user that those buttons are related to the information below. In pages where there are sections of information we added dividers, the name of the

section followed by a line that fills the rest of the width of the element (Figure 4.6). On the "Views" page, as we have pages and templates, we placed the action button at the end of the line (Figure 4.7), so it is still consistent with the rest of the design and easy to understand.

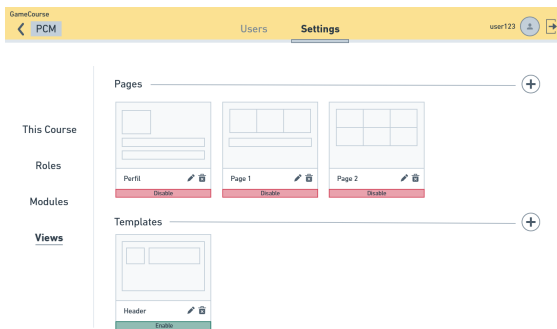


Figure 4.7: Views page.

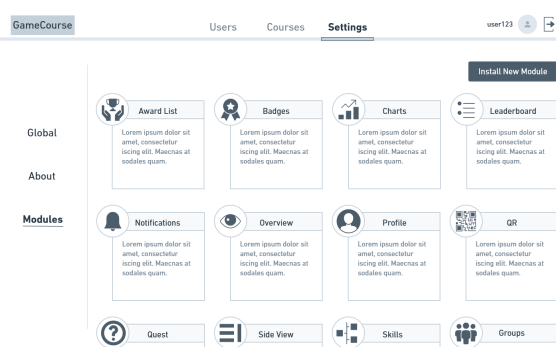


Figure 4.8: Modules page.

4.1.4 Roles page

For the "Roles" page we first used a table to list all the roles and added action buttons to move them up and down in the hierarchy, the ones higher would be above in the hierarchy (Figure 4.9). However that is not how the hierarchy of the roles work. We have the Default role from which three mandatory roles are descendent: Teacher, Student, Watcher. We can create more roles at the level of these three or create descendants of them. In another words, it does not matter the order in which they are but instead it is the layer level where they are. Not having a better idea we kept the original logic of design and used the layers and drag and drop system. The moving buttons were changed to three vertical dots as this is the convention. To remove pages with only one detail we incorporated the landing page definition on the line of each role (Figure 4.10). To make sure the user is in total control of this page, besides the "Save Changes" button we added undo/redo icons to allow the user to go back one or multiple actions. We also added a section for feedback from the system after a save action.

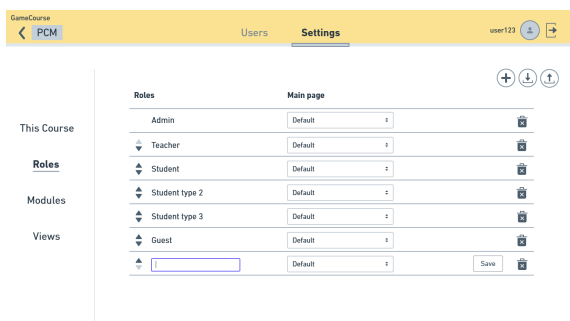


Figure 4.9: First version of the Roles page.

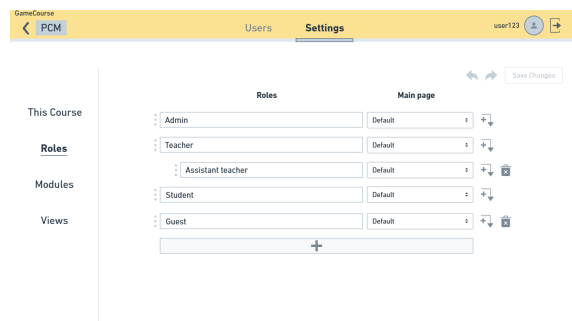


Figure 4.10: Final version of the Roles page.

4.1.5 Login page

Another page that every user sees regardless of being an admin or not, is the "Login" page. As another team will be adding more login options to the GameCourse system, the system can not go directly to the Fenix login page. For that we created a simple page with the GameCourse logo on top followed by the available login methods represented by their logo, so the user can chose his/her desired login method (Figure 4.11).



Figure 4.11: Login page.

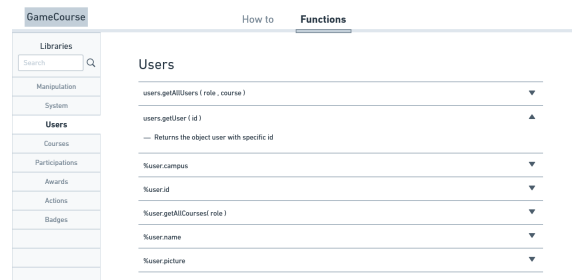


Figure 4.12: Functions list on documentation.

4.1.6 Documentation

On the documentation pages, we added a sidebar to quickly guide the user between the available sections on the "How to" page and the available libraries on the "Functions" page. In this case, we did not use our previous sidebar look. Instead, we went for the tab style due to the resemblance that a documentation web page has to a physical folder of documents divided into sections by tabs (Figure 4.12). Inside each section, the text content was kept the same, but we needed to restyle the functions list. On the initial version of the system, this would be represented by a list of boxes with the function name and its description. To make it easier for the user to find the desired function, we chose to show only the function's name, and on a click, show its description, using an accordion mechanism. For the look of it, we came up with two possible scenarios:

- Use a plus/minus icons before the function name and have no visual separation besides white space between functions;
- Use a down/up triangle at the end of the row of each function created by the lines separating them.

In order to keep the look of the whole system consistent, we chose the second style as it has the same concept as the tables for listing objects.

4.1.7 View editor

All this information on the documentation pages is mainly to help the user use the "View Editor", the easier it is for him/her to find it the better. So we decided to create a slider accessible from every stage of the view editor. This slider not only has a shortcut to the documentation but it also has a tutorial and information about the available libraries.

This page is not a page where the user will come and spend just a couple of seconds, he/she does not need quick access to other setting pages. So we removed the settings sidebar and made the editor full width. From testing the SapLitmos system we knew we wanted to do something similar to make the process of adding new parts faster, so we added a simple box with an add icon in the center, that will be visible inside every part that allows a new one to be included. To fix the issue with the toolbar buttons that are too small and difficult to select, we made the toolbar bigger and fixed it at the bottom of the page. Every time a part is selected its toolbar will show up at the bottom of the screen.

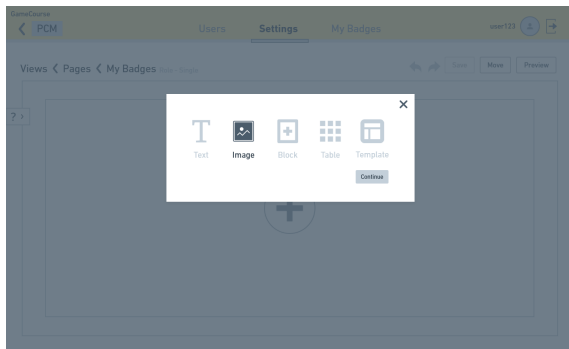


Figure 4.13: Add a new part in the View Editor.

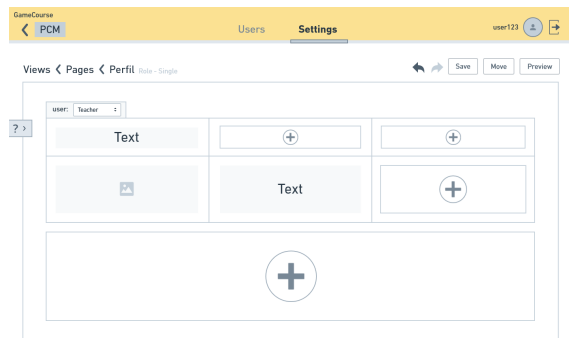


Figure 4.14: Change aspect of a part.

The modal for adding a new part became mainly visual. Instead of selecting the type of part from a dropdown, we select the icon that represents it (Figure 4.13). On the edit part modal we changed the title "DOM Manipulations", some users had trouble understanding the meaning of the section because it is a technical term, so we put "Loop Data". On this section of the view editor we also added a tutorial to better explain each field, where the user is capable of moving forwards and backwards. To ensure the user does not forget he/she is on a tutorial we also added a top bar saying "Tutorial Mode".

One of the goals of Views on the system is to allow different variations of the same page for some type of users. That is achieved through aspects that, on the initial version, can be selected before entering the view editor. We wanted to remove that middle step and incorporate the change within the view editor, so we added a new option on the toolbar to add a new aspect version to the selected part. Once a part has more than one aspect associated, we add a dropdown just above the part, to select the target aspects and configure its version (Figure 4.14).

4.2 Low Fidelity Prototype

To create a LFP where some tasks could be executed and the collection of screens would look like a system we used a prototyping tool called Marvel². This online tool allows us to select a particular area of an image and link it to another image, creating the effect that a website would have. From the wireframes on whimsical we exported an example of each page as image to use on the LFP. However to simulate interaction on the final LFP³ we needed to create a list of available tasks, that would go through all

²<https://marvelapp.com/>

³try the prototype here: <https://marvelapp.com/prototype/6ifjg5>

the main pages, and create the screens for each step of the interaction. On that list (Appendix G), we included tasks like:

- Activate the user João Costa.
- Delete the course "Nome" with 56 students.
- On the course PCM enable the module QR.
- On the course PCM add a new page called "My Badges", role type: single, enable and edit it.
- On the course PCM edit the page "Perfil".

4.3 Heuristic Evaluation

Before the development phase, we decided to make a formative user study with experts in UI & UX to evaluate the Low Fidelity Prototype, using a Heuristic Evaluation based on Nielsen's 10 Usability Heuristics [26]. This type of intermediate evaluation allows an iterative process, where design problems are quickly found and fixed before any development, which prevents unnecessary development work and saves time.

4.3.1 Participants

The number and type of Usability experts was chosen based on the investigations of J. Nielsen [26] where he tested the influence of the level of expertise on the difficulty of the problems found, and the influence of the number of experts on the amount of problems found. He found that having varied levels of expertise among the chosen experts is beneficial because the same person will not always be the best evaluator, and some of the hardest-to-find usability problems will not always be found by who found the most problems. About the amount of evaluators he found that for each added expert new problems will be found, but that this benefit will be lower at each new added expert. For that reason he recommends an ideal number of five experts, that will find 75% of the problems, and minimum of three for around 60%. Having that in mind, we requested the presence of two designers with a specialization in Usability and UX and three teachers of computer engineering in the Information and Visualization area. Even though we only needed five experts, we also included two computer science students, who had recently finished a course about Usability and UX, as they were available at the time.

Level	Description	Necessity to fix
0	No agreement whether this is a usability problem or its severity	
1	Cosmetic problem only	Does not need to be fixed unless extra time is available
2	Minor usability problem	Fixing this should be given low priority
3	Major usability problem	Important to fix, so should be given high priority
4	Usability catastrophe	Imperative to fix this before product can be released

Table 4.2: Levels of severity of a usability problem

4.3.2 Procedure

To perform the Heuristic evaluation we had the option to do the experiment with every expert and then gather them for a discussion, but to save time on the expert side we decided to include an observer [26], that after each experiment would collect all the problems found and later combine all the information.

Each experiment was done on an isolated room, with no interference from the outside, where only the observer and the expert were present. The observer started by presenting the project and informing the expert how the experiment would go. Then each expert was asked to perform the list of tasks defined on the LFP (Chapter 4.2) to get the feeling of the system and to make sure no page is forgotten. After having completed the last task, the expert was given two sheets: one with the 10 heuristics of Nielsen along with a small description of each of them and the process to classify the severity of a usability problem (Table 4.2); and another where he would be able to describe each problem found, the heuristic that it broke and the severity of the problem. Then he was given free access to the LFP in order to better analyse the whole system and to identify the problems.

4.3.3 Results

On the Heuristic Evaluation 57 usability problems were found (Appendix E). From those, more than 70% were declared as level 2 or 1 on the severity scale, meaning they are minor usability problems or cosmetic ones. Only four had the highest level of severity, which required immediate attention and fix on the design (Table 4.3). Even though one problem can not have two different levels of severity, it can affect more than one heuristic. When looking at the data grouped by Heuristic we notice that the most affected one is H4-”Consistency and Standards” with 15 problems mentioned, followed by H2-”Match between system and real world” with 14 problems reported (Table 4.4). The least affected heuristic was H9-”Help users recognize, diagnose, and recover from errors” with only three problems being reported.

	Level 1	Level 2	Level 3	Level 4	Total
Problems found	16	25	12	4	57

Table 4.3: Number of problems found by level of severity.

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Problems found	4	14	5	15	7	11	7	11	3	4
%Level1	0%	7%	40%	40%	14%	36%	43%	55%	0%	0%
%Level2	50%	43%	40%	33%	29%	27%	43%	18%	67%	75%
%Level3	25%	36%	0%	20%	43%	36%	14%	18%	33%	25%
%Level4	25%	14%	20%	7%	14%	0%	0%	9%	0%	0%

Table 4.4: Number of problems found by heuristic as well as their presence in each level of severity.

One of the problems with level 4 of severity was related to H1, H2, and H5, regarding the verb tense of some labels, like ”enable” which translated an action instead of a state and could confuse users. Two other problems were related to the aspect of buttons: the button to add a role to a user was not looking

like the other buttons, violating H4; and the moving buttons on the Roles page which did not transmit the information that the main roles (Teacher, Student and Watcher) can not be moved, violating H2.

Some of the problems reported with severity of level 3, related to H2, were about missing attributes on both the Course and User objects, including "Visible" and "Nickname" correspondingly. Even though we tried to distinguish the navbar options inside of a course from outside, it was still reported, as H4 problem, that it was confusing having "Settings" twice with different contents. Also related to the navbar, it was reported as level 2 of severity and violating H6 that the user should be able to see what is inside the settings before entering that section.

As level 2 of severity, it was also reported the lack of confirmation/success messages after adding a new item, which is a clear violation of the H1-"Visibility of system status". As well as the change of location of the help button on the View editor, once the edit part modal is open, violation H4. Another problem for this level of severity was related to the H3 and H7, in which it was reported the lack of search abilities on the Modules and Views pages, reducing the level of freedom of the user and the flexibility of the system. Another H4 problem, this time on level 1 of severity, was regarding the size of the whole system. As it was designed with no real reference of font size, one of Whimsical's limitations, the wireframes and prototype got too big and needed to be sized down.

4.4 Discussion

The Heuristic Evaluation allowed us to detect a lot of problems on the design that affect the Usability of the system. From the 57 problems we were able to fix 29 of them, some directly on the Wireframes⁴ for reference during development, like label fixing, and others done only during development as it would take too much time to correct it on the wireframes, like the size down.

We fixed all the problems reported as level 4 of severity. Added a new dropdown to the navbar when hovering the settings option, to let the user know what pages he/she can find inside and potentially save one click, and changed the label to "Course Settings" when inside a course. A search box was added at the top of the Modules and Views pages, along with a success and error message section, that was also added to the remaining listing pages.

All the buttons reported as inconsistent were analysed and changed to better match the look of the system. Missing attributes were added to the listing pages and correspondents "add" and "edit" modals. And the help button was moved to be always at the left side of the page. Even though we had more problems being reported on the View editor, they designed was not altered so the problems could be later analysed when developing that section of the system.

⁴final version of the wireframes: <https://whimsical.com/tese-Tv7J6qNoKvSuyuPwY5ZME9>

Chapter 5

Development

In this chapter we will explain the second phase of the project regarding the development of the new GUI for the GameCourse system. We will go through each section and domain of the project in chronological order of events. One of the technologies used during the development was AngularJS, as it takes care of the pages interaction through states, controllers and scopes. On states we defined the routing of the system, in other words, which pages are available to navigate and through which urls. On the controllers we generated the HTML structure of the pages and made the needed requests to the API to obtain the desired data. This data was stored on the scope, manipulated through scope functions, and associated to GUI elements, allowing automatic re-renders when the associated information changes.

5.1 Structure and style of main components

When we work on top of something that is already so complex, like GameCourse, it can be difficult to not let new code be affected by old one. There are other components on the same page that might affect the functionality of the new one. We may end up using existing classes on our components and have to make completely different code to support or override the existing one. In order to avoid all that and save time fixing problems, we began our development phase by creating the structure and style of our main components separately. To make this process faster we used Codepen¹, an online code editor that besides giving us a blank sheet, shows multiple parts of our code (HTML, CSS and JavaScript) on the same window along side a preview of the result. There, we were able to define²:

- Navbar;
- Sidebar for the documentation;
- Sidebar for search, filter and order section;
- Table for item listing;
- Modals, including a modal for inputs and other for verification;
- Section dividers;
- Cards for both modules and courses/views;

This step does not mean that the style and structure of these components was totally done at this phase and that they did not suffer any change after. It only means it worked as a guide and base for the page's look, so we could focus more on the logic part of each page.

¹<https://codepen.io/>

²<https://codepen.io/collection/XRdMQd>

5.2 New GameCourse structure

The first step inside the GamecCourse was organize the system so it would have the page organization that we defined on the beginning of the design phase. We started by removing the list of modules from the global settings of a course to place it on a new settings page called "Modules", having just moved the content we only needed to create a new state and new controller. On the system domain we created a similar page called "Installed Modules" where we also placed the list of modules, but without the enable/disable section, which made our new API function much simpler, as it only had to collect all modules and their information from the *ModuleLoader*.

Then we moved onto the Users, on system domain the users list and mechanism to add and remove admin permission was in the settings. As nothing else was on that page, we simply changed the location of the state to be outside of the settings, removed the page link form the settings side bar and added it to the navbar. Inside a course the listing of users is divided between several pages, one for students, one for teachers and user roles are managed on the roles page. Therefore, we removed the first two pages and reused their code to create a new page with a list of all user. On the existing API function we added an option to receive a request for all roles and changed the current request on the controller to match that change. Finally we created a new state for this page and added the its link to the navbar. As we were changing the navbar we took the opportunity to change the option "Settings" to "Course Settings", so it is easier to distinguish system settings with course settings.

Regarding the Courses we also had the actions spread over multiple pages, a "Courses" page outside the settings to list the courses and allow access to them, and another page inside the settings to allow the actions: add, delete and deactivate/activate. We removed the last one and merged its content into the first one, having now all the actions on a single page, accessible from the navbar.

The last section in need of relocation was the Roles. We had their hierarchy placed on one page and then one page per role to define the corresponding landing page. We started by merging all these pages into one, to be able to have all the landing pages together, and removed the unnecessary pages from the settings sidebar. While doing that, we changed the function *getRoles* on the class *Course* in order to return all the role information instead of just the name and created a new API function to gather data of all the roles and submit landing page changes. After having this temporary page working, we placed its content inside the existing "Roles" page, leaving all things related to Roles in one page.

5.2.1 Code refactoring

After the first contact with the system, we realised the files were all too long and mixed multiple concepts, going against the principles of programming: Separation of Concerns (SoC) [27]. We started by dividing states and controller declarations creating folders for each of them, on the *js* folder. To avoid having everything in one file the states were still divided into two files for settings and non settings pages, and the controllers separated between "system" and "course" domain and the object to which they are

related. Besides that, we still created a new `.js` file to place all the auxiliary functions that are used multiple times around the code. Regarding the API functions, they were all written on a single file, `inpho.php`, so we created a new folder at the root of the project to divide them into different files regarding the domain and object to which they are related.

5.3 Navbar

As planned during the design process, on the navbar we added two extra sections, one for the Game-Course logo and another for the username, profile icon and the logout button. To have the username being displayed, we had to add the user information to the scope of the page, which is done over the controller of the whole system. It is also there where we changed the menu to include the possibility of dropdowns. To the existing `mainNavigation`, we added the attribute `class` to all items, so we can then add a specific class to each option, and the attribute `children` to let the system know if it is a dropdown option. We then created `settingsNavigation` where we added all the options available in the settings sidebar, to create the dropdown options and quickly let the user know of all his/her options. On the navbar structure, we added a verification connected to the scope to evaluate if the option being added has the attribute `children` true so we can then render the settings dropdown.

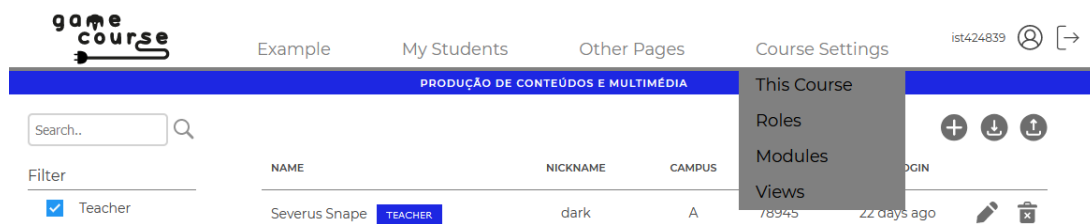


Figure 5.1: Navbar inside a course with open dropdown on settings

Since the menu content changes when we enter a specific course we rewrote the function that registers the menu on the Core class to include the new attributes and created a new function to add options to the settings dropdown. Not only can the content change but it can also grow. Inside a course we are able to create Pages and as long as the View they contain is of role type single, they will be added to the navbar. If we add too many pages on a course, the navbar will have no space for all the options, which leads to a scalability problem. To fix this we created an algorithm that will check the size of the navbar and if it is bigger than expected will move some of the options into an "Other Pages" dropdown menu (Figure 5.2). This algorithm is ran every time the page is resized, reloaded and when we enter or exit a course. On the resize case, to take into account the possibility of having more space after the action, we first reset the navbar, placing the options that were already on the "Other Pages" dropdown back to the main menu and just then use the algorithm.

Last but not least, upon entering a course, to visually identify it, we add a small bar under the navbar with the course color telling its name (Figure 5.1). To make sure the name of the course is always visible we created a function that will check how dark is the color and put the letters in white or gray.

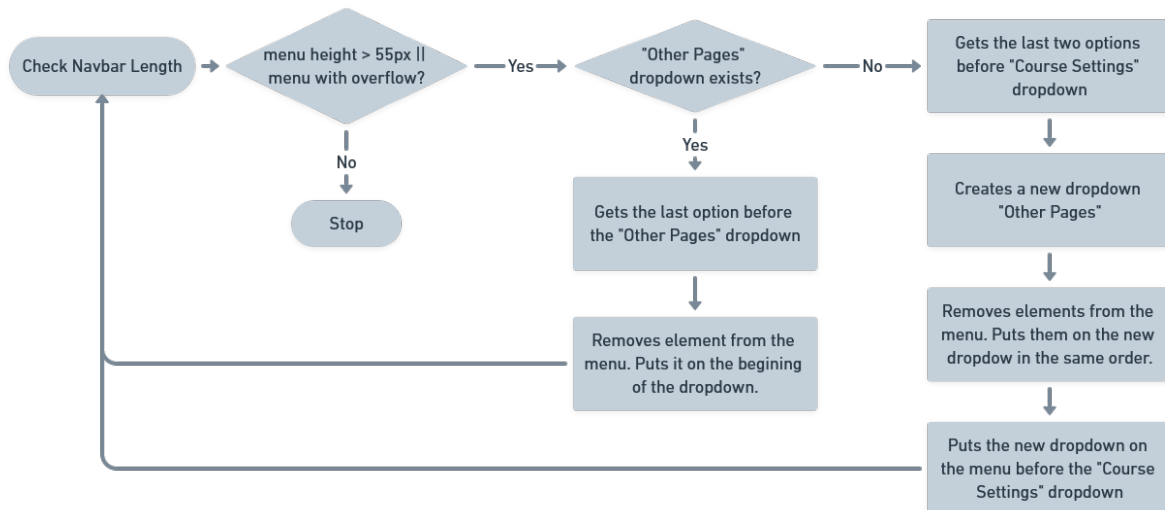


Figure 5.2: Flowchart of the checkNavbarLength algorithm

5.4 Courses page

The courses page can be accessed by everyone to either select a course to enter or to maintain any global information of a course. While users with admin permissions will be able to execute both actions, users with no admin permissions will only be able to execute the first one. Because of that we created two different versions of this page. We have already brought the actions that were on the settings page to this one but we still have to add missing information, add the ability to edit a course and the search, filter and order functionality.

5.4.1 Class preparation

To be able to maintain a more complex and scalable system, the existing fields (*name*, *isActive* and *isVisible*) were not enough to define a course. So, we started by changing the database structure related to this class by adding three more fields:

- color, that will visually identify the course;
- short, as it is already usual for courses to be referenced by their short name;
- year, to keep track of the same course (that keeps the same name and short) throughout the years.

Next, we changed the "new course" function to include information from all fields, including the *isActive* and *isVisible* that previously were being set with default values, only the *name* was being received as input. Then we created an edit function that receives information about all the attributes and changes the existing object on the database.

Finally to be able to later make the on/off buttons functionality we needed functions to get and save information about the *isActive* and *isVisible* attributes. As the first one was already being manipulated we only needed to add the *isVisible* functions.

5.4.2 Admin version

For the admin version, we started by building the table where the courses information will be displayed (Figure 5.3). All the columns have their content centered and a correspondent label on top, except for the name column that has its content aligned to the left and the color and action columns that do not have a label on top. The inside of the table is built using the scope, which allows us to define the structure of one line and repeat it for each item on an array. This array is the list of courses that is received from the API request. This API function is the same for both versions of the page, but it was changed to return different sets of courses depending on the user's authorization. For admins, it returns all the courses of the system with all their information and still adds the number of students that each course has.

NAME	SHORT	# STUDENTS	YEAR	ACTIVE	VISIBLE	
teste	t	0	2015-2016	<input type="checkbox"/>	<input type="checkbox"/>	
Visualização de Informação	VI	0	2020-2021	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Teste para cron	tpc	96	2015-2016	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Produção de Conteúdos e Multimédia	PCM	4	2020-2021	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Conceção Centrada no Utilizador	CCU	3	2017-2018	<input type="checkbox"/>	<input type="checkbox"/>	
Análise de Dados	AD	2	2020-2021	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Analise sistematica de dados - Copy	ASD	0	2016-2017	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Analise sistematica de dados	ASD	1	2016-2017	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Figure 5.3: Courses page for admin users.

In order to make the on/off buttons work on the Visible and Active columns (Figure 5.3) we created two functions on the controller that will read the state of the button once it is clicked, check to which course it is assigned and request a change of value to the API. There the system will check if the selected course exists and use the save function we did before, if it does not exist it will return an error. On the controller, after the request, if everything goes as expected, the attribute is updated on the correspondent course object on the scope array. However if an error is returned a modal will be created to show it. We added this verification on every API request of the system and created a function that will generate the modal with the error message, that in terms of looks is very similar to the verification modals we design but has only one grey button saying "Confirm".

To add a new course we created a modal with all the inputs needed to define a course and focused the user action on the first input for quicker filling. Even though we choose during the design not to use labels, to simplify the look of the modal, we were able to add a floating label that will show inside the input at the right side once the user fills it with information. This way we still get the intended look and give the user information about each input when information is already there, especially useful when

editing an object. For the color input we added a preview square to show the selected color inside the input, and added an external library that allows us to create a color picker. The color picker will update the input value and the color shown on the square preview (Figure 5.4). To prevent errors from the user we created a function that checks if all mandatory fields are filled and disables the save button if not. Once a request for saving the new course is triggered the controller will send the scope variable, that kept all the inputs information, to the API. For the API function we decided to use the one that already existed and add the missing fields, as this function was already ready to the duplicate action.

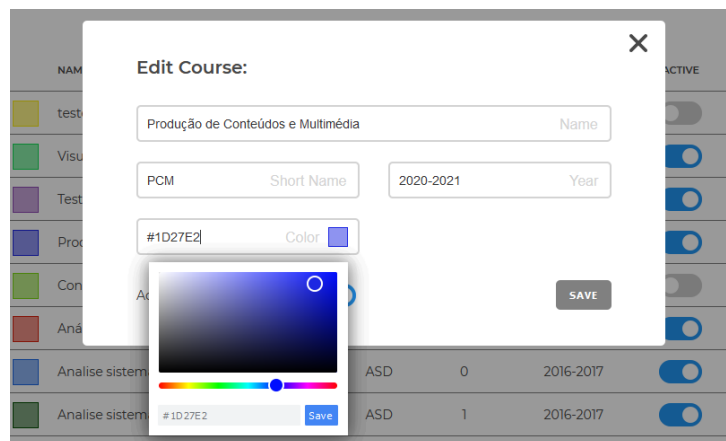


Figure 5.4: Add a new course modal with color picker.

Previously the duplication action was done through selecting a course to copy from during the create course action. We kept the mechanism on the API side which includes sending the *creationMode* defined as "similar" but on the controller and user side we made a "one click" action, where the user can change from the table the course object he/she wants do duplicate. All the information of that course is copied and sent on the API request, but the name is changed so it include " - Copy" at the end (Figure 5.3).

Another action available to each item on the table is the edit. Just like in the "add" action we created a modal with all the inputs and associated a scope variable. Once a course is chosen to edit, by clicking on its correspondent icon, all its information is passed onto the variable and the inputs are populated (Figure 5.4). This action has the same verification as the "add" one, so if the user tries to save a course with an empty input the button to save will be disabled. On the API we created a new function that will receive all the course information including its id and through the function we created on the Course class it will update the course info.

Delete a course was already possible to do in the initial version of the GameCourse, so what we did was replace the existing browser warning, that forced the user to write "DELETE", for a verification modal where the user is reminded of which course he/she selected and given the option to proceed and delete it or to cancel (Figure 5.5).

After any of the previous actions a success message is displayed on the top of the page for 3 seconds and the list of courses on the scope is updated by making a new request to the API to get that data. This keeps the information updated on the user side without the need for a reload of the page. Finally

to make it easier for the user to navigate on a big list of courses we made the header of the table fixed to be always visible as we scroll through the items and we added an hover effect on the lines to easily associate the icons with the right item.

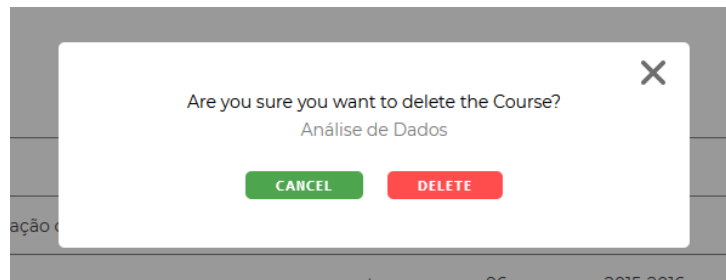


Figure 5.5: Verification modal before deleting a course.

5.4.3 Search, filter and order algorithm

Finding a particular item among a big list of items can be difficult and time-consuming. So to make it easier we created a sidebar for search, filter and order by a particular attribute (Figure 5.3). As we intended to use this sidebar on both versions of this page and on the users page we created a function that creates the sidebar structure. This function must receive two arrays, one with the labels for the options on the filter sections and another for the options on order by section. It will start by creating the search section with an input text box and a magnifying glass icon, then it will verify if the arrays provided have at least one option and if not that section will not be created. On the filter section, for each option it will create a checkbox input with the label. Whereas on the order section it will create a radio button and two arrows to determine if it is ascending or descending order. To each option it will associate an id generated from the label so it can later be used on the algorithm.

To make the algorithm work we started by duplicating the scope's array that keeps the list of items, one is used to keep all the items and the other to display the selected ones. As we want to be able to complement both actions that reduce the amount of items displayed, search and filter, every time one of them is triggered both of the mechanisms will be ran, first the search and then the filter. Both will act on the array of items for display purpose, that will be restore beforehand with a copy of the one that has all the items. In this way we are able to apply a specific filter, have the set X as result, then apply a search and get the set Y and go back, deleting the search and still have the same X set as expected.

The **search mechanism** will be triggered each time the user types on the search box. It starts by validating the input, as it will not do anything if it is empty or a combination of spaces. Then for each item it will verify if its attributes contain the input value, on the courses case it will search in the name, short and year attributes. If the item has a match in any of its attributes it will be added to a temporary array of items, that after analysing every item, will be check to see if there was any match. In case of no matches a warning message will be displayed to let the user know of the result, but if this has at least one item, the array that is used to display the selected items will be replaced by a copy of this one, leaving only the matched items displayed.

On the **filter mechanism**, the trigger occurs once any of the checkboxes is clicked. It will start by checking the state of each option and see if the selected combination is possible, as at least one of the options of each pair needs to be selected. For this page the pairs are active/inactive and visible/invisible. In case one of the pairs has no option selected a warning will be displayed to let the user know it needs to select at least one of the options. Next, for each item, it will check if it matches the selected conditions and save it on a temporary array that later will be copied onto the display array. Just like on the search, if the temporary array is empty, a warning message will be displayed to let the user know there were no matches found for the filter applied.

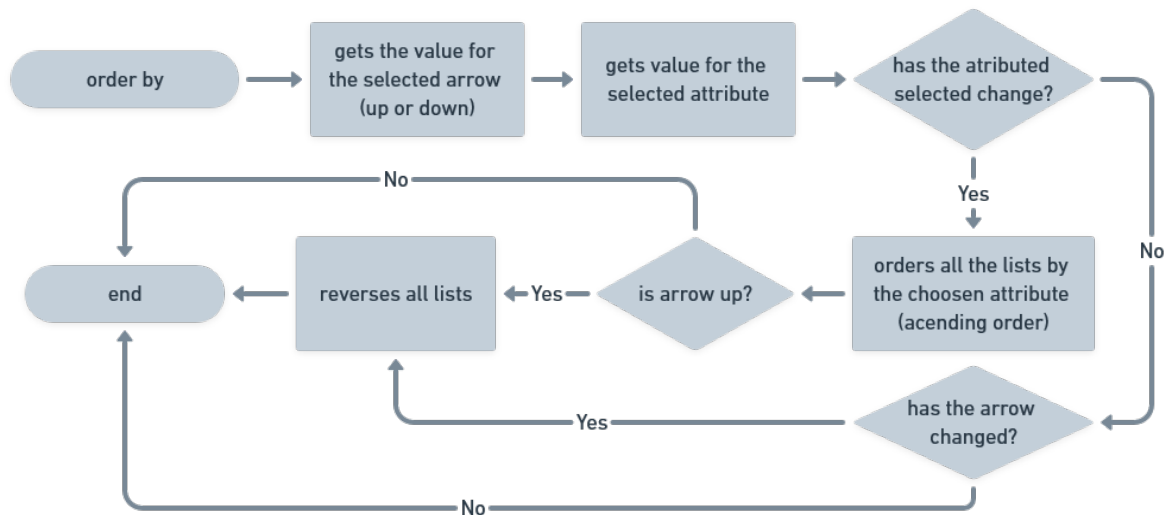


Figure 5.6: Flowchart of order mechanism

As only one option can be selected at a time, the **order mechanism** keeps track of the previous selected attribute and direction of order. Once a click occurs either on the options or on the arrow icons, it will collect the current state (Figure 5.6). If the selected attribute is still the same, it checks the arrows, if they changed it reverses all the arrays with items, if not it means nothing changed so nothing happens. If the attributed selected is not the same, it starts by applying a sort on all the arrays with items using an auxiliary function for that specific attribute. This sort is done on ascending order, arrow down, so then it checks the selected arrow and if it is up, descending order, it reverses the arrays. As we apply this over all the items arrays the user will immediately see the change on the page.

5.4.4 Non admin version

For the non admin version of this page, the API function will return only a set of visible courses where the user is registered. To display them we created two sections of cards: the first one for the courses that are currently active, since they will be accessed more frequently; and the second for those which are not active anymore (Figure 5.7). For this type of user the only information about the course that he/she can see is the short name and the year, placed on the label of the card, and the name of the

course that appears on hover on the colorful part of the card. Having that in mind we limited the side bar to the search and order actions that go through these three attributes. In a search the algorithm will go through all the courses on both sections separately and give a warning message if no match is found.

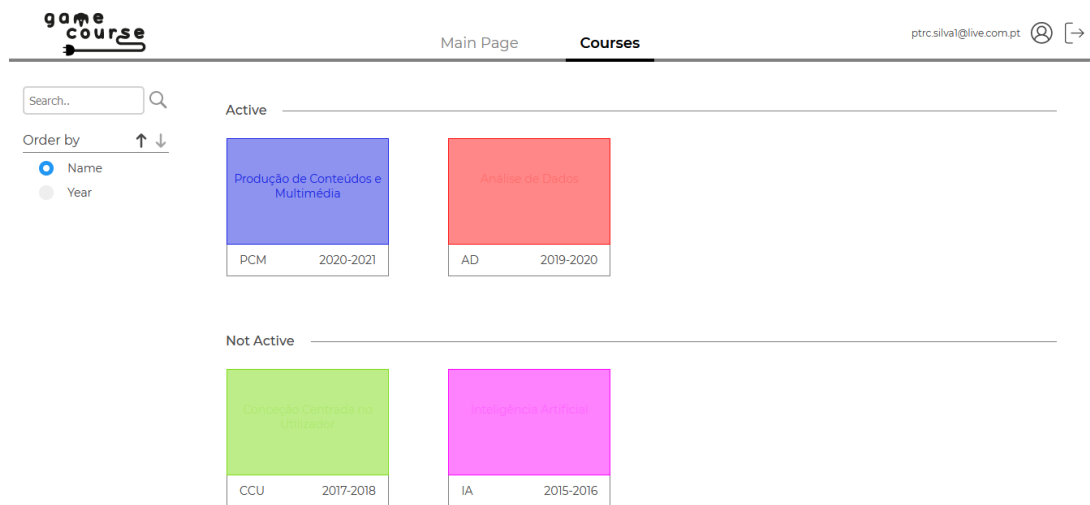


Figure 5.7: Courses page for non admin users.

5.5 Users page system and course domain

The Users page did not exist previously on the system domain. There was no way of adding a new user without having to connect him/her to an existing course and if some information needed to be updated it would have to be done through that course. The only way of checking all the users on the system was through the admin management on the settings. Inside a course the users were divided by roles on different pages and the only place where all of course users would be together was on the Roles page where we could change their roles but not their information. We have already brought the list of users to a new page outside settings, on both the system and course domain, but all the functionality is missing.

5.5.1 Class preparation

Just like in the Courses case a lot of information is missing from the User class. We started by adding three new attributes to the class the database: *isActive*, *nickname* and *studentNumber*. Even though we already had the student number information stored, this was being saved on the *id* making it difficult to alter if this value was placed wrong at first. The *id* was changed to be of auto increment value and the *studentNumber* was added as unique since this is still a way of identifying a student outside the GameCourse. Because of this change we created a new function that selects the user object by providing the student number and warned the other team working on the project about this change.

The edit and add functions were changed in order to contain the new attributes and the *isActive*, that was set with a default value before. Then we created a delete function to allow the total removal of a user and another function that calculates the last login of the user. Although we do not have that

type of information on the User class we have *lastActivity* on the CourseUser class that contains a time frame that is updated on every activity of the user inside a course. The system starts by getting all the CourseUser objects related to the user and calculate the most recent *lastActivity* value. Then it transforms the time stamp into a string that tells how long ago was that interaction, getting results like: 2 minutes ago, 1 week ago or never in case the user has not interacted in neither of his/her courses yet.

As we want to display some of the User class information on the Users page inside a course and the CourseUser class extends the User one, we added a *get* and *set* method for each User attribute we need to access through the CourseUser.

5.5.2 Users page - system domain

The structure and functionality of the Users page is very similar to the Courses page, we use a table to display all items with the options to edit and delete, we have a sidebar for search, filter and order by an attribute and we have the action buttons that let us add a new user (Figure 5.8). To create the sidebar we sent the pairs Active/Inactive and Admin/NonAdmin for the filtering and the rest of the attributes for the ordering. On the API we changed the function that returns the users information to include the list of courses the user is registered in, how many they are and the user last login calculated through the function we created on the User class.

On this list of items we also have on/off buttons for the attributes *isActive* and *isAdmin* so two new API functions were created to allow a quick change of this values. To make all API calls safer and prevent errors from expert users that might use the command line to change the system items, we verify if the user has admin permissions and if the items selected for change in fact exists, if not an error message is returned to let the user know of his/her mistake.

NAME	NICKNAME	STUDENT Nº	# COURSES	LAST LOGIN	ADMIN	ACTIVE
Yu Cheng		97282	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Yone Linda Cnatingius		97638	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Viviana De Brito Bernardo		87709	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Victor Manuel Gomes Ribeiro		97075	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Vicente Maria Pereira Cândia Canas Simões		84776	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Téva Christian Nils PAQUIN		97648	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Tyrion Lannister	drinks	95674	0	never	<input type="checkbox"/>	<input type="checkbox"/>
Torstein Lundervold Nesheim		94778	1	2 days ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Tomás Alves		75541	1	1 day ago	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 5.8: Users page on the system domain.

To add a new user we created a modal that has all the inputs for the user attributes and an extra input of type file to add a user profile image. By clicking on this input the user is allowed to choose a file

from his/her computer that represents an image. Once an image is selected a preview will be displayed on the input box (Figure 5.9), if the user wants to select another image he/she can simply click again to select another one. Just like on the other input modals the controller will check if all the mandatory fields are filled before letting the user save his/her action. Another verification is done on the API side for this action. As the student number must be unique, before creating the new user, the system verifies if there is any other user with the same student number. If there is already one the API returns an error that is displayed on the page through an error modal letting the user know about his/her mistake. In case everything is valid the user is created and the image is added to the system folder "photos".

For the edit action we created a modal that looks just like the add one, but has an extra section to display the user list of courses (Figure 5.9). In case the selected user already has a profile image the preview will be automatically filled with it, but again it can still be changed by clicking. On the API side we added a new function where it verifies first if the student number has been changed and second if it is already taken, only then does it update the user item and photo. To allow the delete of an user we also added an API function for that and a verification modal referring the user that is being deleted.

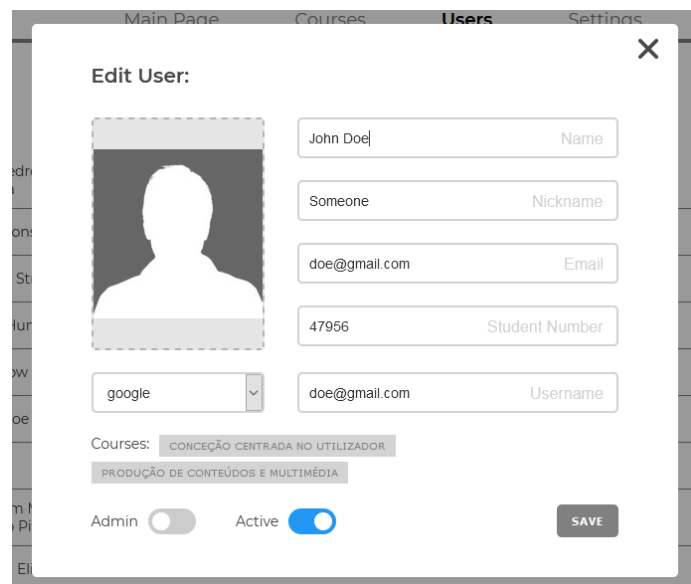


Figure 5.9: Edit User modal on the system domain.

5.5.3 Users page - course domain

This page is very similar with the previous one but has some changes to include course related information. On the controller we start by making a request to a new API function that returns the name of all roles of the course. The resulting list is used to give the options for the filtering section of the sidebar that in this page does not have to verify pairs of options. We are then able to make a request to the API function that returns the course users information to build the table. In this function besides adding the new attributes of the *User* class we added the *CourseUser* roles. These are displayed using role tags in front of the user name on the first column of the table. To match the course identification bar this tags are painted on the course color.

As in the previous page we created an API function to remove the user from the course and added a verification modal. When we were building the add new modal and the API function we realised there was something missing that would have a great impact on the usability of the system. Until this point, we had never thought about the option of adding an existing user to the course. We quickly created two more versions of this modal on Whimsical³:

- select, where the user chooses if he/she wants to create a new user or if he/she wants to select a user that is already registered on the system;
- add new, with all the input fields necessary to create a new user and course user;
- add existing, where the user is presented with a list of users that are not in the course, from which he/she can select as many as he/she wants and give a specific role to all of them.

The same modal is used in all three versions and its content is changed depending on the user decisions. If at any time the modal is closed it is restored to the selecting mode (Figure 5.10). For the add new version we added all the inputs that we had on the Users page over the system domain plus a *campus* one that is exclusive to the *CourseUser* class and a section to add roles. On this section the user is able to select a role from a dropdown, that lists all the roles on the course, and add it to the new user by clicking on the add button, which will add a role tag to the roles list below it. To remove a previously selected role the user just needs to click on the role tag, as it will appear a delete icon while hovering on it. On each of these actions the controller will re-render this section so the dropdown has only available the roles that were not selected before. On the validation function that controls the save button of the modal we added a verification to only allow the save when the user has at least one role selected. In order to save this new user, on the API, we created a new function where we first try to create a new *User* just like when doing it on the system domain, then create the *CourseUser* that will connect the user to the course and if everything succeeds we add the selected roles to the user. The same look and behavior is applied to the edit modal.

For the third version of the modal we started by making a new API function that will return the list of users that are not registered on the course by comparing all users of the system with users of the course. In terms of look, on this modal we added:

- search box, where the user can type the user's name or student number he/she is looking for;
- listing box, where every item is separated by a horizontal line, like on the tables, and a add icon is placed on the end of the line;
- dropdown with the available roles on the course, so the user can select which role to associate with the selected users.

The listing box is populated with the result of the API request, and each item is represented by the user's name and student number (Figure 5.11). By clicking on the add icon of an item a tag will be added to the search box with the name of the selected user and the user will be removed from the list. If the user made a mistake and wants to remove the selected user, just like when selecting a role, he/she just

³<https://whimsical.com/tese-Tv7J6qNoKvSuyuPwY5ZME9>

needs to click on the correspondent tag. The user can add as many system users he/she wants and by searching the list below will be reduced to match the search. Just like the other modals the save button here will only be available when at least one user is selected and a role is chosen. Finally to submit this action we created a new API function that will do all the verifications needed when asked from a command line, valid course and existing system user(s), add the selected user to the course by creating a new *CourseUser* and associating the chosen role.

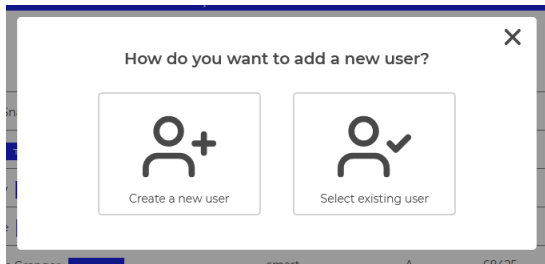


Figure 5.10: Select mode of the add user modal.

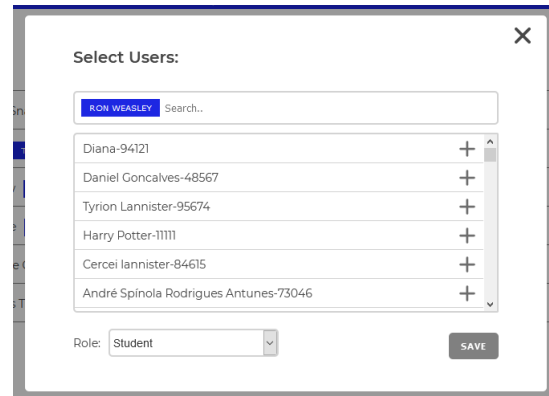


Figure 5.11: Add existing mode of the add user modal.

5.6 Settings page

For the settings page both on the system and course domain, we removed all the sidebar options that were on a second layer, as well as the "Configuration" option inside the course, and applied the already prepared style. To keep the consistency with the other pages we made the content section scrollable so the rest, sidebar and navbar, would be fixed and always visible. On the more simple pages we changed the content to include our section dividers but kept the information as it was.

5.7 Roles page

The Roles page is meant to manage the roles of the course, their hierarchy and the landing page associated. Having that in mind we started by removing the section where we could associate a role to a user, as we already added that functionality to the Users page.

During the design phase we decided to keep the existing mechanism of the roles hierarchy manipulation, done through Nestable⁴. It not only allows us to drag and drop elements and nest them inside each others but it also converts its content into an array which is what we use to save the roles hierarchy on the Data Base. However both the Nestable mechanism and the functionality that saves this information, had some problems that needed to be fixed and adjustments to include the landing page manipulation.

In order to incorporate the landing page association we changed the API function that returns the Role hierarchy to also return the pages of the course and all the information about each role. Then on

⁴<https://github.com/dbushell/Nestable>

the controller we changed the structure of the page to include a second section per Nestable item, where we put a dropdown with all the available pages, associated on the scope to the correspondent role. After applying the new style we moved the save button to the top right corner of the page as the other action buttons of the system (Figure 5.12). All of these had to be done with caution as the mechanism has to be aware of the structure, using its exclusive classes, so it can drag and drop all the items.

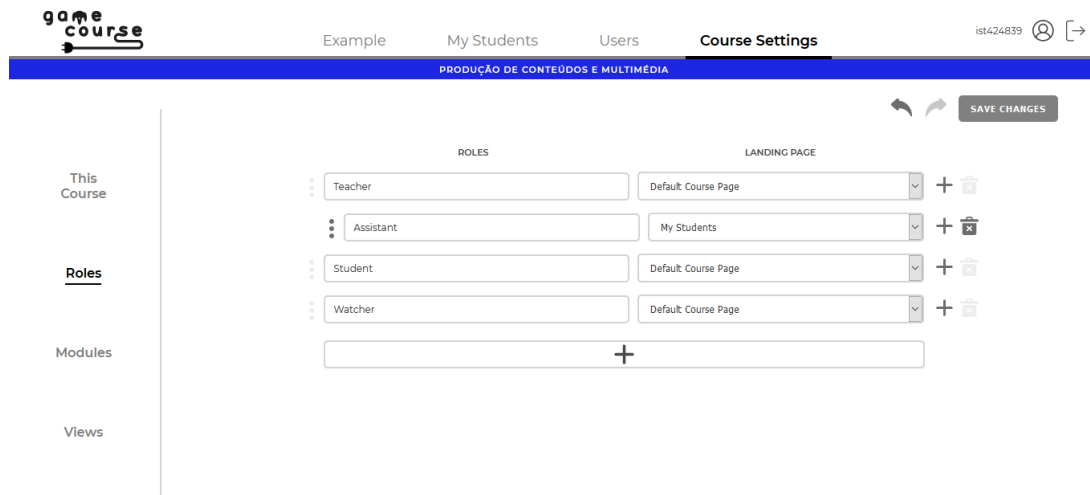


Figure 5.12: Roles page with the roles hierarchy.

On the Nestable mechanism we started by removing the functionality that allows collapsing behavior of each nested item and fixed the space created at the beginning of each item, depending on the layer of hierarchy. We then changed the icons to match the ones we already used and informed the mechanism of the use of the *icon* class. As our icons are not fully opaque, we noticed that each time a Nestable item is moved, the mechanism adds another add button at the end of the row. This was happening because the mechanism does not initially add an add button if the item is on the last authorized layer, but when moved to higher layers it needs the button. We fixed the problem by adding a verification that checks if an add button is already there or not, as the mechanism already checked the layer.

Even though we had already changed each item to be displayed with an extra field for the landing page on the controller, the mechanism was not aware of that and each new item added was created to keep the old structure. To fix that we sent the list of pages on the Nestable creation and added the dropdown everytime a new item is created. At the same time on the controller we add the association of the landing page value to the scope. While doing this changed we came across another problem, the new role is added through a prompt and if the action is canceled a new item is still added with empty values. We started by exchanging the prompt by a simple modal with one field, for the role name, and added a verification on both the Nestable and controller side so they only register the new role if the user submits it. As we always refer to the roles by their names we also added a verification to the save button that disables it when the input name is already taken by another role.

The roles *Teacher*, *Student* and *Watcher* are the base roles of a course and they must not be deleted

or put under each others on the hierarchy. To prevent the user from moving these roles, we associated the Nestable class *dd-nodrag* to the correspondent items and blocked the delete by not associating the correspondent action to the icon, when creating the Nestable items on the controller. To visually inform the user these actions were disabled we lowered the opacity of the icons and changed the cursor to blocked while hovering them (Figure 5.12). Contrary to the other pages, here the delete button does not trigger a modal to confirm the action, since it will only be applied once the user saves changes.

At this point the information about the roles, the hierarchy, and the landing pages was being stored in multiple places. So before sending the information on the API request to save, we merged the information so we have the roles full information in one object and the hierarchy on another. While we did not changed this API function we did fix some issues on the *Course* function *setRoles* that saves the information on the database. During several experiments, we noticed that when saving, the roles already registered were being deleted and created again, leading to an id problem on the association user-role inside the courses. We fixed this by also sending the id of the roles on the API request, and verifying it on the *setRoles* function. If the role id is empty it means it is a new role and we add it to the database, otherwise we update the role information rather than deleting and creating again. To still delete the roles that were not kept, we verify if their id is among the ids of the submitted roles. Once the action is completed a success message is displayed on the top of the page, like in the previous pages.

5.7.1 Undo/Redo mechanism & State Manager

Roles are delicate information, as they may change the users authorizations and the way they see the system both due to the landing page definition and to the Views aspects. To make the editing of this information easier we added undo and redo functionalities to the page, so the admin user can go back to verify something or to cancel an action. We started by creating the State Manager (SM), which keeps three arrays: one for past states, one for current state, and the third for future states, and created the functions needed for the undo/redo functionality:

- **New State:** the SM will move the current state to the end of the past states, replace the current state for the submitted one and clean the future states.
- **Undo:** the SM will move the current state to the end of the future states and the last state of the past states into the current state.
- **Redo:** the SM will move the current state to the end of the past states and the last state of the future states into the current state.
- **Can Undo:** verifies if the past states has any state.
- **Can Redo:** verifies if the future states has any state.

As this mechanism is not dependent on the state content it can later be used on new pages. In this case we create a new state with all the roles information and hierarchy, whenever a move or delete is done or a landing page is changed. This new state had to be a value copy of the existing arrays and inner information as keeping the references would alter previous states as well. The confirmation

functions are used to activate the undo and redo buttons placed near the save button, which trigger their correspondent SM functions when active. After either the actions a state is received from the SM and the page is re-rendered to match the now current information.

5.8 Modules pages

For the modules we have two pages, one to display all the installed modules over the settings on the system domain, and another over the settings of each course to display all the modules and allow enabling/disabling actions. To be able to apply the style we designed, we started by adding a new attribute called *description* to the *Module* class and incorporate it on the *ModuleLoader* functionality. Then we went to all existing modules to add the new attribute, with an adequate description of the module, onto the its registration function and added an *icon.svg* file in the module's folder with an representative icon. Finally to bring this information to the page, we added the description attribute on both API functions that return the modules information.

On both pages we created a card for each module, where we render the module's icon on the circle, the name on top of the card and the description below. Inside a course we also place a status bar to immediately tell the user if the module is enabled (Figure 5.13). We used a grid schema to display all the cards, making the page responsive to multiple widths, having always the same distance between cards and aligning the fourth element of a row with the end on the content section. Since the content of both this pages can scale a little and the user might want to find a particular module, we added a search box on top and used the same mechanism of search we used on the Courses and Users pages, by verifying if the search matches either the name or description of the modules.

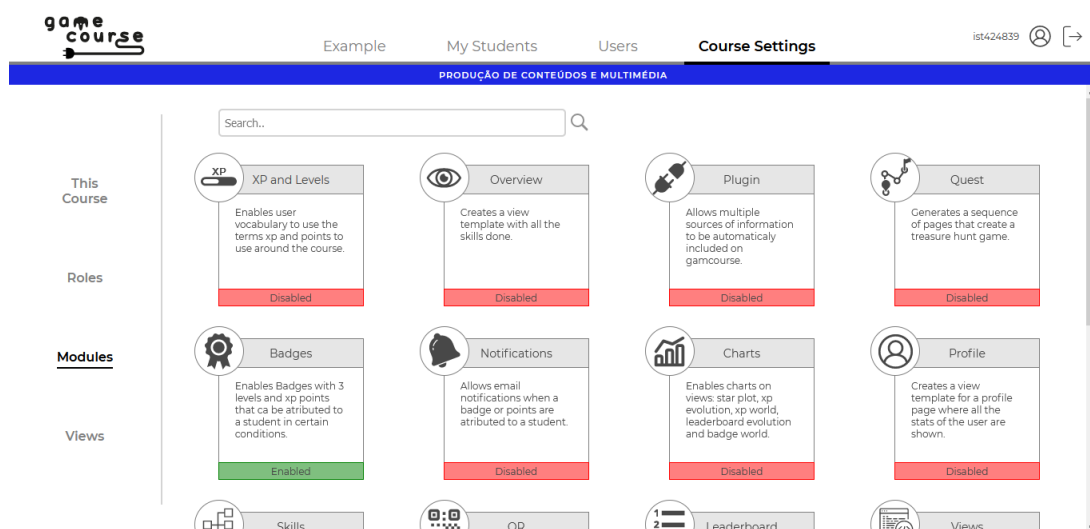


Figure 5.13: Modules page inside a course.

As we want to be able to enable and disable a module inside a course, for this page we added an extra modal, open after clicking on the card. There all the module information (icon, description, name, version, path, and dependencies) is shown alongside an on/off button to enable/disable it and a save

button to submit the action. This last button is only enabled when the status of the first button changes, implying an activation or deactivation of the module (Figure 5.15). In order to prevent user errors while trying to enable a modules that has its dependencies not enabled, we altered the API function that returns the modules information to also return the state of each of its dependencies and if it can be enabled or not. With this information the system disables the on/off button if the module can not be enabled and associates the color green and red to the dependencies that are enabled and disabled correspondingly (Figure 5.14).

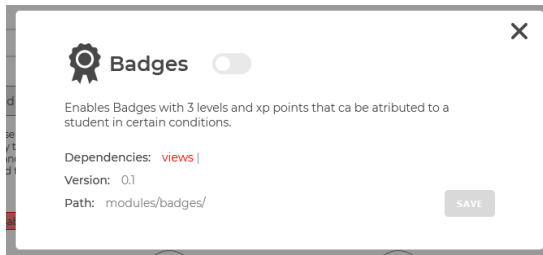


Figure 5.14: Module modal with disable on/off button.

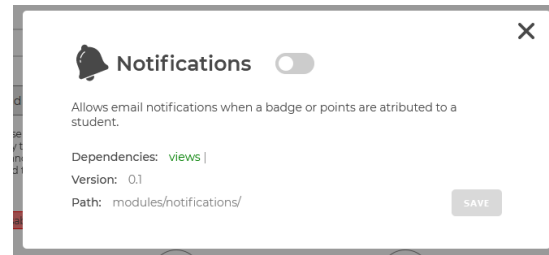


Figure 5.15: Module modal with enable on/off button.

5.9 New Configuration System for Modules

Before this project, to be able to populate the database with information regarding the modules, we would have to use external scripts and run them on the machine where the system would be installed. To reduce the amount of effort and time spent on these tasks, we created a configuration system available to all current and future system modules. This system is based on blocks and not only saves time to the Teacher or Admin that is going to use it to configure their course, but also to the developers of modules who don't need to create a whole new page for their module configuration.

5.9.1 Code Restructure

In order to make the system work we had to make sure every module was working as intended. If modules have pages that are supposed to only be accessed once they are enabled, those pages can only be declared (using states) inside a piece of code that is added when the modules are in fact enabled by the user. Some modules had this declaration outside the intended place, which made these pages accessible through URL on courses where the correspondent modules were disabled. We fixed it by placing the pages declaration inside the module activation code.

5.9.2 Configuration page

The goal of the configuration system was to be able to have a generic configuration page, that depending on the selected module would render their information and needed inputs for configuration. Having that in mind we created a new page with three possible sections:

- General inputs: to display all the variables that define and change the behavior of the module.
- Listing items: for modules like Badges, that have inner objects that can be created, edited and deleted. This section works and looks very much like the other listing pages, with a table and options to manipulate the items.
- Personalized section: sometimes neither of the previous sections are enough or a variable is too complex to a single input field, so we added a personalized section where the developer can create what he/she wants.

To make this work we created a unique controller to manage and create the configuration page. It starts by asking the API if the selected module has any of the three sections, and if yes what are the needed inputs. Then it will generate each of the sections, using the style we had already chosen for sections and inputs, and when needed call the API again to save new information. In the general inputs section the save is made through a save button and in the listing item section it will save at each action: create, edit and delete confirmation. For the personalized section the saving is established by the developer of the module. On the API side it will create an object of the selected module and request information about these three sections, first if they have it and then get or give information. In case of the first request (get information) it will return the gathered information at once to the controller. In case of the second request (save information) it will give the module the information sent by the controller. Even if the module's developer forgets to tell whether or not he/she wants a configuration page or each of the sections, this system will still work because the mother class, Module, has the needed information to reject the building of the page and each section if needed.

These two parts of the configuration system make the page exist by itself. However it is a set of functions per module, with default values at mother class, that provide the content of the page. The first function *is_configurable* is to tell if the module has a configuration page, if true a "configure" button will be displayed on the module modal at the Modules page, to lead to the configuration page. Besides redefining this function to return true on the new module class, the developer has to create the state for the page with our controller and add a parameter with the id of the module, otherwise the generic controller will not know which module it is.

To build each section of the configuration page we have a "has", "get" and "save" function that will tell if the correspondent section is going to be needed, what information is needed and to save information coming from the user inputs. The "has" functions work as verification, only returning true or false values, and the other ones have specific rules in what they must return ("get" function) and in what they receive.

In the **general inputs section** our controller was built so the module could have several types of inputs (Figure 5.16):

- text: for regular text fields where the user can write anything;
- date: for a time related input;
- on_off button: for Boolean values;
- select: to create a dropdown with a set of available options;

- color: to use a color picker just like the one used on the course add/edit modal;
- number: for fields where the input must always be a number;
- paragraph: if the field needs to take a lot of information.

Figure 5.16: Available types on general inputs section.

For that to happen the system has to have information about the fields it needs to create. The function *get_general_inputs* provides that information, here the developer receives the id of the course where the module is being configured and must return a list with all the inputs he/she wants to create. Each input is defined by an array containing the following pairs key-value:

- name: to be displayed as label of the input;
- id: to internally identify the input and to be able to later collect information;
- type: selecting one of the mentioned available types;
- options: in case of having selected the "select" type of input, the value here must be an array with the options for the dropdown;
- current_val: to be set as initial value on the input.

After the save button is pressed on the GUI an array will be sent to the *save_general_inputs* function, with several sets of key-value where the key is the id of the input, defined on the get function, and the value is the input coming from the user. Inside the function the developer only needs to access the array by the right id and tell the module what to do with the information. Both functions receive the course id so the developer is able to access database information to fill the current_val value and save the new information.

The **Listing items section** is more complex than the previous one as the system needs to know the items to be displayed, which inputs are needed for the add and edit modals, the headers for the table

and the ids of the attributes to be displayed on the table. Because of that the *get_listing_items* function needs to return an array with the following pairs key-value:

- *listName*: to give a name to the section;
- *itemName*: to be used as header of the modals;
- *header*: to set the labels of the columns of the table, it needs to be an array;
- *displayAttributes*: to be able to access the items attributes, it needs to be an array and to be on the same order as the header;
- *items*: the set of items to be managed;
- *allAttributes*: to define the inputs on the add/edit modal, follows the same syntax as the inputs of the previous section, but does not need the key *current_val*. Here only the types text, select, number, date and on_off button are available.

Once an add, edit or delete action is made, the *save_listing_item* function will receive information regarding the type of action done and the item involved. In case it is an add, it will receive an array with the key-value pairs defined on the inputs (attribute id and input value), if it is an edit it will also receive the id of the item, but if it is a delete the id is the only information sent about the item. Inside the function the developer only needs to evaluate the action type and tell the module what to do in each occasion. For the same reason as before, both functions also receive the course id.

The **personalized section** was created to support more complex cases and even though it is simple on our configuration system it leaves more work for the developer. He/she must visually create the section and define the API and module functions to save the information. In order for this section to be built the *get_personalized_function* must be declared and return a string with the name of the module function that visually creates the section. This function will be then called on the controller and the following variables will be sent as arguments: *scope* in order to save page related information; *element* where any new HTML structure must be added; *smartboards*, to be able to call an API function defined on the module; and finally the *compile* so it is possible to relate scope information to HTML structure. In this way the module function will be implemented just like a controller but will be integrated on the existing one with other sections.

5.9.3 Documentation

All the information above was added to the modules documentation page, on a new section called "Module Configuration" for easy access. It also includes code examples for all the cases (Appendix F).

5.9.4 Badges configuration page

To prove and demonstrate the value of this system we defined a Badges configuration page. As described previously, we started by creating the state to be able to access the configuration page and redefining the *is_configurable* function to return true. The module Badges has one numeric attribute

called Max Reward that needs to be changed by the user. The general inputs section allows exactly that, so we redefined the *has_general_inputs* function to return true in order for this section to be created. Then we redefined the *get_general_inputs* function to return an array with the max reward input. For that we choose a number type input and set the array key *current_val* to the current value of the attribute through an existing get function. To save the value we redefined the *save_general_inputs* function so it would call the existing saving function of the max reward attribute.

For Badges we also needed to manage the list of badges so we used the listing item section. First we redefined the *has_listing_items* function to return true. Then, as the badge item has a lot of attributes, we chose a set of them to be displayed on the table, created the needed arrays regarding that information, collected the existing items from an existing function and defined all the inputs needed to create and edit a badge. In the *get_listing_items* function returned these values on the supposed key-value pairs together with the "listName" set as Badges and "itemName" as badge. In order to save the actions made to items, on the *save_listing_item* we just verified the type of action and called the existing function for that action. The result is a full functional configuration page (Figure 5.17), that without this system would have to be completely done from zero.

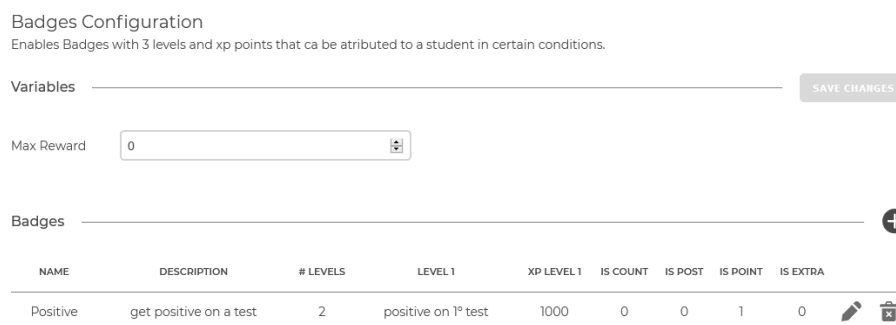


Figure 5.17: Badges configuration page.

5.10 Views page

On the Views page we already had sections to separate pages from templates. As all sections were already styled, we started by moving the add button from each section, to the end of the line of the section and replacing it by the add icon (Figure 5.18). Then we changed the structure of each item so they would be represented by cards instead of plain text, similar to the ones used to represent the courses when viewed by a non admin user. On the left side of the card description we placed the view name and id. On the right the icons of the tasks available for that item, delete for Pages and globalize, export and delete for templates. On the top of the card we placed a picture of the structure of the page as background and the edit icon visible while hovering it. All this process is managed by the controller with the use of the scope as we did in the rest of the listing pages.

The add action in both types of views, already triggered a modal so we updated its style and structure to match the rest of the system modals. For the delete action we replaced the existing prompt for a verification modal mentioning the name and id of the selected view. On the templates case we added an extra warning on the modal message, to let the user know that the delete might affect other pages that used the selected template. We also added another modal that is shown after the export of a template to inform the user to where he/she can find the exported file. As on any other listing page we added a search box so the user can quickly find the view he/she wants (Figure 5.18). This search looks at the view names and places a warning on top of each section if no match is found within that section.

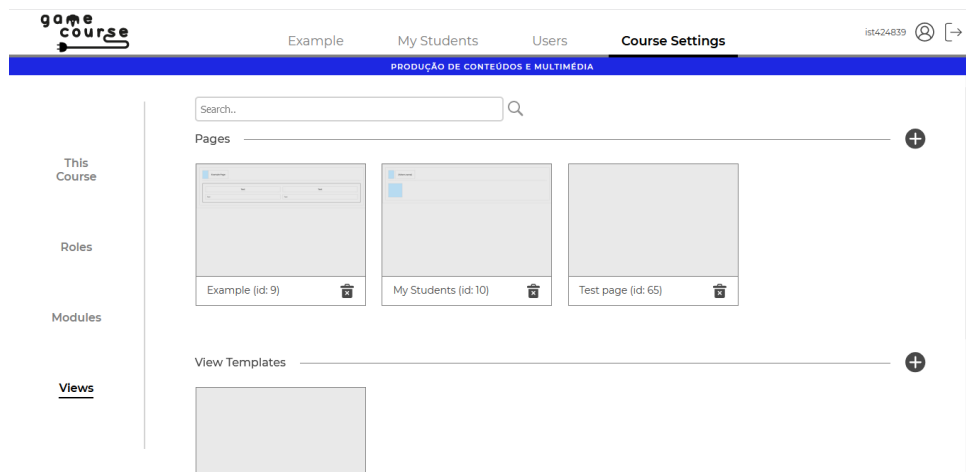


Figure 5.18: Views page.

During the design phase we planned to incorporate the aspect selection inside the View Editor and to do some technical changes. However, due to time constraints, we ended up only visually updating the View Editor and applying smaller changes that were reported as critical during the first User tests (Chapter 3.3). On the aspect selection page we added two sections, one for details of the chosen View, telling the view name as well as type, and another to incorporate the selection (Figure 5.19). We altered the buttons and inputs so they would be consistent with the rest of the system and gave the list of options the same look we used to display the available users to add to a course. Finally we changed the label of each list to include the variables that the selection of the aspect is related to, so the user has a quick remind of what values these Expression Language variables will have.

5.10.1 View Editor

Inside the View Editor we started by removing the sidebar and recreating the breadcrumb so the user can click on it to go back to the listing page. Then we changed the action buttons to match the other pages in look as well as positioning, and exchanged the "Undo" and "Redo" buttons by the icons already used on the Roles Page (Figure 5.20). These buttons were previously configured to only show when the correspondent actions are available, but it was changed so they would have an active and inactive version as the rest of the icon buttons of the system. On the "Save" button we added a small functionality in which it triggers a screen shot of the View to be shown on its view card on the listing page.

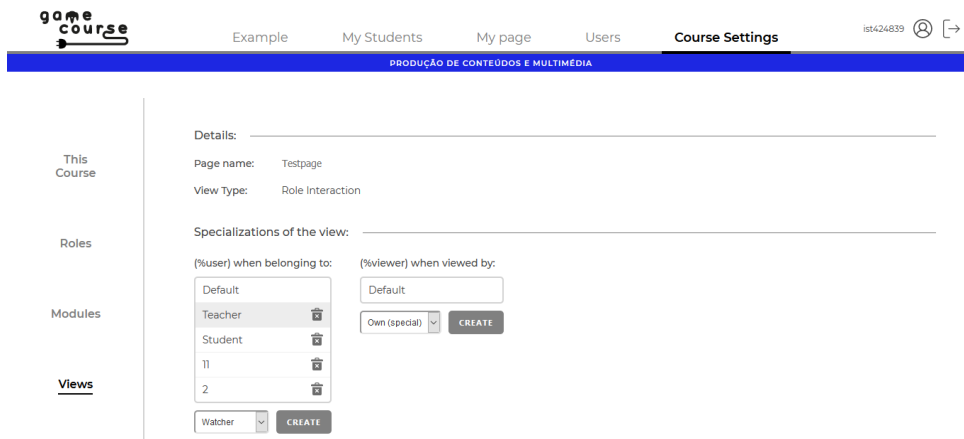


Figure 5.19: Intermediate page for aspect selection.

Another change on the functionality of the View Editor is related to the toolbars of each View Part. Instead of using the hover, with *mouseenter* and *mouseleave* events to show the correspondent toolbar, we changed it to be a click event with selected and not selected states. Once a Part is selected all the others go to the not selected state and the toolbar is shown, clicking on the same Part twice also makes the state transition, closing the toolbar. This allowed us to not only move the toolbar to the bottom of the page (Figure 5.20), since we can access it outside the visual space of the selected Part, but also to make its options bigger, which was not possible before because it could cover other View Parts. All the icons of the toolbar were also changed to match the rest of the system.

In order to keep some distancing between the multiple Parts and to be able to identify where each one starts and ends we added white space between and inside all the Parts and added a light grey border, dark when selected. For those who can have more Parts inside, tables and blocks, we made the corners rounded and for the others, texts and images, we made the corners squared (Figure 5.20).

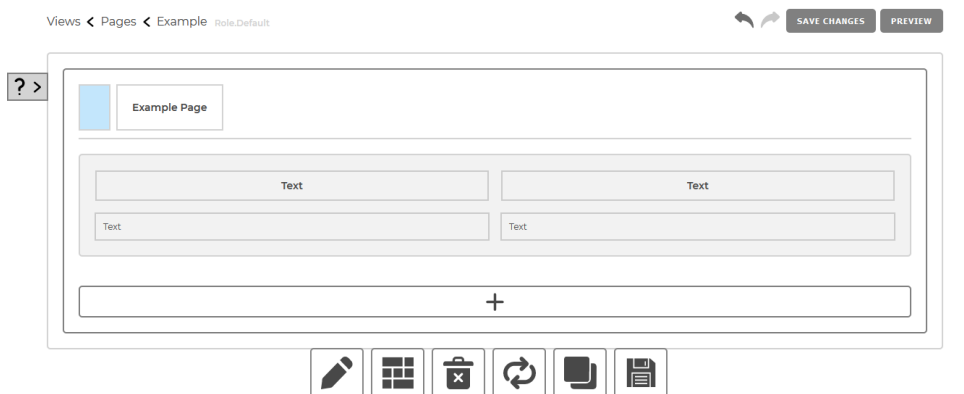


Figure 5.20: View Editor with a block on edit layout mode.

Even though we did not add a constant add button we still incorporated that button into the existing functionality. When selecting the "Edit Layout" option on the toolbar a wide button with a plus icon is added at the end of the View, Block or Template (Figure 5.20). The existing dropdown options were

moved into a new add modal that is opened by clicking on the add button. Although the option of "Edit Layout" also exists on the Table Part, here it will show the table manipulation options to add headers, columns and rows. These new options also use a toolbar but here they were kept on their original place since they are related to the existing columns and rows (Figure 5.21).

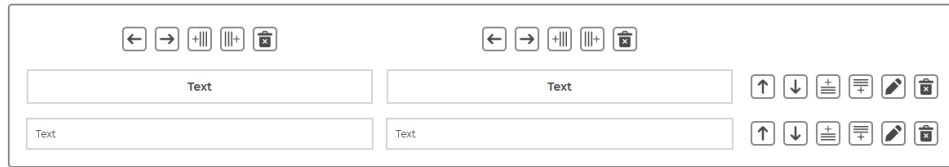


Figure 5.21: Table Part on edit layout mode.

On the **Add Part modal**, instead of using two dropdowns we created a selection using images, representatives of the multiple Parts, as well as labels (Figure 5.23). Only by selecting the Template option will a dropdown appear with the list of the existing templates. In this case we also used a on/off button to let the user chose between using the template as a reference or as a copy. On the controller we reused the existing logic, and after clicking the Save button the result of the on/off button will determine which method is used to add the template.

Any Part can be switched by another through the toolbar that will open the **Switch Part modal**. On the initial version of the system, there was already a modal for this task using a simple dropdown. We changed it to look like the Add Part modal, but without the on/off button to add a template by reference, since this option is not available. One thing missing on the previous version of this modal was the information of the type of the selected Part, forcing the user to remember it. So, we identified the type of the selected Part by adding another label saying "Current" and by disabling it (Figure 5.24).

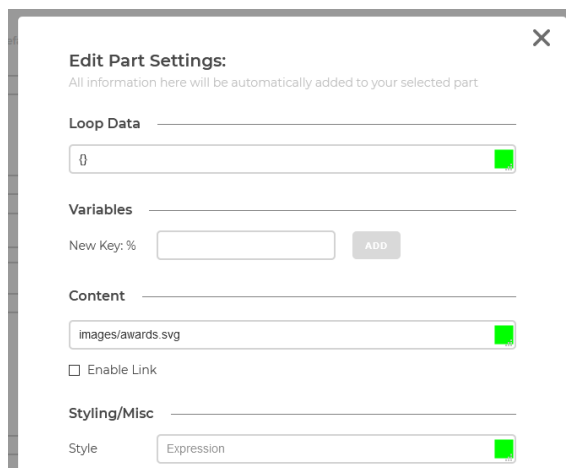


Figure 5.22: Top part of the Edit Part modal.

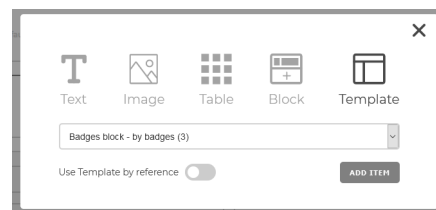


Figure 5.23: Add Part modal.

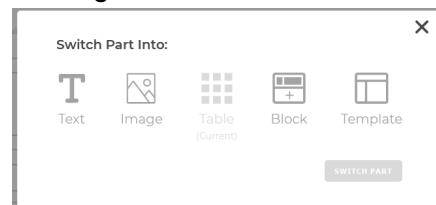


Figure 5.24: Switch Part modal.

On the toolbar of every Part there were two very similar options: saving as template and saving as template by reference. We decided to incorporate both actions on one single option with a **save as template modal** and distinguished them by an on/off button. Just like in the Add Part modal, over the

controller the value of this button determines which method is used allowing us to reuse the existing functionalities. During some tests we noticed that the save using references had an issue as it would replace the chosen part by the last known template instead of the one just created. This was happening because the function that visually replaces the Part by a template by reference was not waiting for the API return and assumed that the list of templates would already be updated. We fixed this issue by returning the id of the new created template on the API function and by only calling the second function after updating the list of templates and having the id returned by the API.

The edit option of the toolbar also already had a modal so we started by changing its style to match our modals. Then we changed each step of the **edit part modal** to use a section and reorganized them (Figure 5.22). In some cases not all the sections are needed so instead of keeping them with empty content we hide them. We also hide parts of the content when it is not needed, just like in the case of the input that is only used when the visibility of the Part is chosen through a condition. For the sections "Content" and "Events" we replaced the delete buttons by our delete icon but kept the text Add button, since we kept them as text when associated with an input (Add Role case). All the old documentation shortcuts were removed from the input fields as our last change on the View Editor was adding the side helper that is always visible and accessible.

5.11 Documentation Pages

The documentation is the helper of the system but in order for it to really help it needs to be easy to use. To create the desired look we started by dividing the existing content into topics and create a title for each. All the topic titles were arranged on a side menu, our tabs, and the correspondent information by sections on the content part. To make the tabs work we identified each tab option and section of content with a specific id and saved the relation. By clicking on a tab the system will search by its pair content, close the other sections and open the selected one. All the documentation pages use this mechanism and define their own version of the function *addClickEvents* to define the relation tab-content.

On the functions page we had to create a function to create the separation of information, side menu and content, as this page is populated dynamically depending on the selected course. For each library it creates a new tab and a content section with an accordion to display all the correspondent functions and their descriptions. On this accordion clicking on the name of the function will open the correspondent description and close the one previously opened, so it only has one section open at a time.

Even though it was not planned on the design, we created a search box to add more flexibility and to make any search faster. Contrary to previous search boxes this one looks into the content of the page instead of evaluating items. Since each page has a lot of information and performing a search at each input change would make the system slower, it only searches after an enter or a click on the magnifying glass. Before starting the content search the input is validated, as no search will occur in case of an empty input or a combination of spaces.

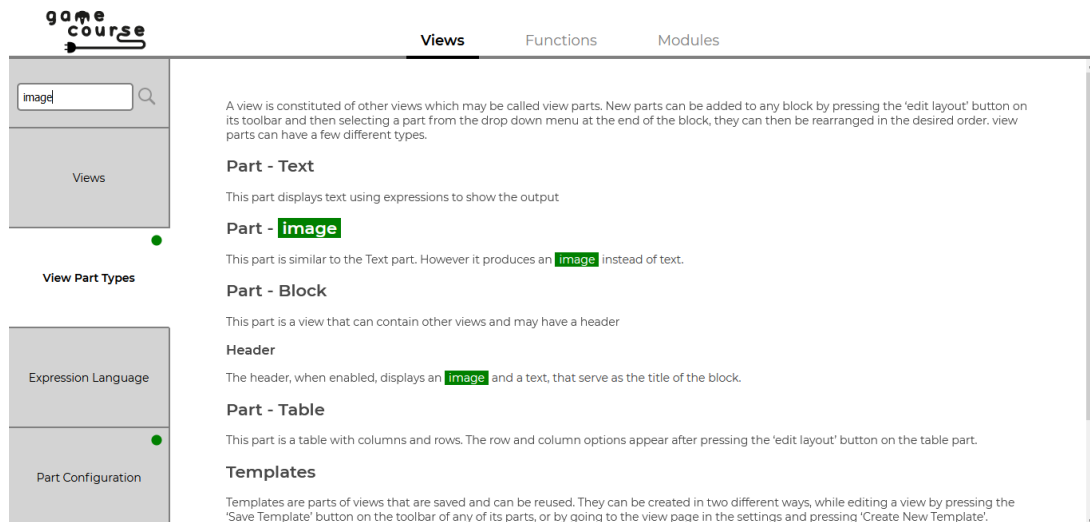


Figure 5.25: Search for "image" on the Views page of the documentation

The search algorithm will start by looking at the content sections as a whole and identify each match with a bright green color on the background (Figure 5.25), similar to the "Control + F" behavior. In order to prevent errors from the users searching for something similar to an HTML tag, like "div" and breaking the page, we changed the structure of the pages so every piece of text is on an element with no children. Having that our algorithm will recursively look into each HTML element, check if it has no children, and if so try to identify a match to the search input. Then it analyses by topic and adds an animated identifier on the tabs that contain a match on the correspondent information section (Figure 5.25). If none of the tabs contain a match a warning "No Match" will be displayed above the search box. The first identified tab will be selected so the user has immediate access to a match.

5.12 Changes to complement new functionalities

While we were working on a new GUI for the GameCourse system, another team was developing new functionalities on the system's back-end side. Those functionalities needed to be available to the user, so we had to make some changes to the GUI and complement the front-end to support them.

5.12.1 Plugin module configuration page

The module's configuration was not the only part of the GameCourse using external scripts to populate the database. There were more scripts, including one to create users based on the information of Fenix. To remove them, the other team created a module called plugin that automatically imports this and other external information into the system. That module needed to be configured, so we had to create a configuration page for it. As this module was done on the initial phase of development of the other team, our configuration system was not yet implemented. Therefore the first version of the configuration page was done on the old GameCourse. After gathering the information about the necessary inputs from

the other team, we created a new controller to create the page and display all the inputs. This page was divided in 4 sections, one for each source of external information: fenix, moodle, class check, and google sheets. In each of them we added a save button to be able to submit information.

We also created a new API function inside the module to get and save information about these four sources. Even though the functions to save the information on the database were not done yet by the other team, we left both the controller and the API function ready to work once those functions were included in the code. To make the development easier for the other team, we left instructions on the document telling what else they needed to do and how to make requests into the database.

Once their development was finished and our configuration system was ready, we transformed the configuration page in order to use the configuration system. As this page has four very particular sections, we used the personalized section, which allowed us to keep the majority of the code already done. The controller was transformed into a function (`pluginPersonalizedConfig`), the state was changed to include the generic configuration page controller and the function `has_personalized_config` was redefined to return true as well as the `get_personalized_function` that now returns "pluginPersonalizedConfig". The last thing to do was to style the page so it would match the rest of the GameCourse system (Figure 5.26), in which we used a grid to be able to place two columns in case of a wider screen and only one on a smaller screen.

Plugin Configuration
Allows multiple sources of information to be automatically included on gamcourse.

Fenix Variables

Fenix Course Id: Nenhum ficheiro selecionado.

Moodle Variables

DB Server:	<input type="text" value="localhost"/>	DB User:	<input type="text" value="root"/>
DB Pass:	<input type="text"/>	DB:	<input type="text" value="moodle"/>
DB Port:	<input type="text" value="3306"/>	Prefix:	<input type="text" value="mdl_"/>
Time:	<input type="text"/>	Course:	<input type="text"/>

Figure 5.26: Plugin configuration page.

5.12.2 Autentication

Initially the GameCourse system only allowed login with a Fenix account, if people outside the institution needed to access the system the only way to add them was by making an exception on the code. Having that into consideration the other team added three more methods of authentication: Facebook, Google, and LinkedIn. Besides creating all the logic involved they also created two new pages, one for method selection and another to give information in case the user has no account created on the system yet.

Having the pages created we only had to style them and adjust the HTML structure where it was needed. We went for a look similar to the one IST website has for authentication. A box on the center with the system logo on the top and the available options below (Figure 5.27). As planned on the design, instead of using a dropdown to select the authentication method, we used the logos of these platforms. Similar to the rest of the system we added an hover effect were the logo goes darker and a circle appears smoothly around it. The same box look was given to the second pages, the information for the user was placed where the login methods were and a button to go back was placed below the box. This button allows the user to go back and select another login method in case he/she selected the wrong one and has a GameCourse account with another authentication method. On the background, behind both boxes, we placed an image of a server from the IST archives.

With this new functionality the class User was changed to have an authentication method and user-name associated. For that reason we changed the User and Course User modals to include this two new fields. As it is a pair of inputs it was easy to place them, we created a new row of inputs just below the profile photo, with a dropdown to select the authentication method and a text field for the username. Then we only needed to change the API function to receive these two fields and send them as arguments on the create and edit functions, as the rest was already done.



Figure 5.27: Login page.

5.12.3 Import and Export

One of the tasks of the other team was to make it possible to import and export the main objects of the system and their related data. It brings not only more flexibility in case of multiple entries but also creates a way of doing a back-up of the system information without having to go into the machine where the system is installed and running. To support this feature we added the import and export buttons to the "Courses", "Users" (both course and system domain), and later to the "Installed Modules" pages. To the import button we associated a modal where we give information about the type of file the system is expecting, together with a file type input and a button to confirm the action (Figure 5.28). However on the export button we trigger the action right away.

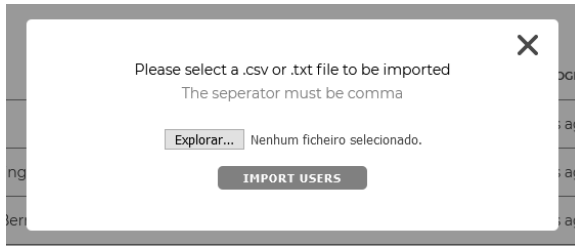


Figure 5.28: Import system users.

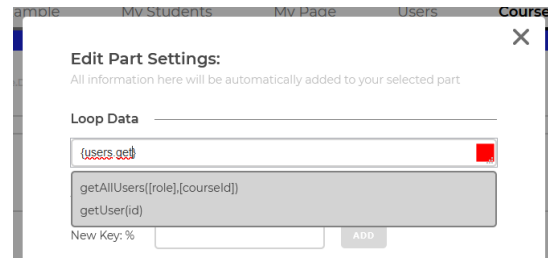


Figure 5.29: Input with suggestion from the autocomplete.

For both "Courses" and "Users" pages, on the correspondent controller, we created a function to convert the information on the selected file into a string that we later send to a new API function. At this point the parsing function, done by the other team to confirm the submitted data, was not ready yet so we only left the base of the API function done: required values and desired response. This response was to inform how many objects were registered on the system and it is used on the target page to give a confirmation message. Also on these pages we added a function to generate a .csv file with the information coming from the API request, to trigger the download for the user. On the "Installed Modules" page we only took care of the import modal, by adding a disclaimer to verify if the imported module follows the rules indicated in the documentation (with a direct link to it).

5.12.4 Autocomplete

The autocomplete feature was one of the features that was planned from the beginning of the design phase. It gives suggestions based on the available functions and libraries, which makes it easier to write the expression language. The other team created the algorithm that checks which input is being written on, evaluates the expression and returns an array with the possible functions available on the expression language dictionary. We created the section to display the options given on the array and added the logic to allow selection with auto fill of the input.

For it to work we start by checking the new options, rendering them on the suggestions box, and associating a click event. First it will check what is the string that triggered the suggestions, if it has one or more dots involved it will only count what comes after the last dot. Then it will compare that string to the suggestion and calculate what is the cursor position for the begin of the string to replace by the chosen suggestion. Just like in a code editor if the user types the beginning of the function it will give him/her the available options and activate autocomplete (Figure 5.29).

5.13 Main page

The Main page already existed and was used to welcome the users to the Gamecourse. It was mainly empty and had no other purpose. We decided to take advantage of this page to add more flexibility to the system through shortcuts. This new version of the page was designed to still welcome the users to the system but to also provide a list of the active courses of the logged in user (Figure 5.30).

In order to obtain the needed information we created a new API function that collects the information about the active courses in which a user is in. Then on the Main page controller we added a request to this function and for each received course, created a link to the course with its full name. As it is still a simple page, all the elements were centered. To prevent clicks on the wrong course we added a hover effect that will make the font bold.

Welcome to the GameCourse system

YOUR ACTIVE COURSES

Produção de Conteúdos e Multimédia
Análise de Dados
Visualização de Informação

My Information



NAME	Patricia
NICKNAME	psilva
STUDENT NUMBER	83536
EMAIL	patricia.i.d.silva@tecnico.ulisboa.pt
AUTHENTICATION	fenix
USERNAME	ist424839

If some of the following information is not right please contact your Teacher to fix it.

Figure 5.30: Content of the Main page with shortcuts.

Figure 5.31: Content of the My information page.

5.14 My Information

A user that is not an admin neither a teacher in a course, has no access to the users page and consequently no way of checking if his/her information is right or not. The "My Information" page is the solution to that problem. This is a simple informative page where the data about the current logged in user is displayed (Figure 5.31).

To make this page work we created a new API function that gathers the information about the object User that is logged in. Then we added a new controller that builds the page and displays the information provided by the API. It uses a flex box to align the profile image, labels and correspondent information columns. These two columns also use a flex box in order to keep the information and its label aligned and evenly spaced. As this page is only informative we added a note at the telling the user that if something is wrong he/she has to contact a teacher in order to fix it. Finally we changed the profile icon on the navbar to a bold version keeping the philosophy of the navbar selection.

Being a profile page it needs to be accessible both inside and out of a course, so we created states for both versions and added their reference to the profile icon on the navbar. Because of the scope we are able to know whether we are inside a course or not and render a different icon, with different links, for each version.

Chapter 6

Evaluation

In this chapter, we will present the evaluation performed on GameCourse to validate the success of this project. For that, we used User Tests, which include the performance of tasks on the old version and the new version of GameCourse.

6.1 User Tests

To validate the effect of the new version of GameCourse, we performed summative user tests, where potential users were asked to perform the same 12 tasks on both systems (plus five more on the new version). To evaluate these tasks regarding effectiveness and efficiency, we collected the time a user needed to perform a task and if he/she succeeded or not. After performing the tasks on each version, the users answered a questionnaire with questions from the Nasa-TLX¹, the SUS², and the User Experience Questionnaire (UEQ)³.

Due to the pandemic situation, it was not possible to perform the tests in person, so we used a software called Zoom⁴. Zoom allows us to create a video call for the experiment, share our screen, and allow interaction from the other side of the call. The main reason we choose this software over others is this ability to allow interaction, so the user could interact with both versions of the systems without doing anything on his/her computer. The new version of the system was installed on a Virtual machine⁵, and the old version was installed on the computer from where the interviewer was making the video call.

6.1.1 Participants

In order to perform a detailed analysis, we conducted 20 user tests (the expected minimum for summative analysis), two women and 18 men. From the total, 11 had previously used the Gamecourse system on the MCP course. However, they had no contact with the back-office part, only visible for admin users. Only one of the users had already fully experimented with the system. All of the users reported previous experience with other LMSs, except for one. All the users were connected to the Computer Science field with nine full-time workers, six students, two working students, and three teachers.

6.1.2 Procedure

Each experiment started with the presentation of the project and a request to record it. Followed by a questionnaire to obtain information about the user, including if he/she had already used the GameCourse

¹<https://humansystems.arc.nasa.gov/groups/TLX/>

²<https://www.usabilitest.com/system-usability-scale>

³<https://www.ueq-online.org/>

⁴<https://zoom.us/>

⁵<https://pcm.rnl.tecnico.ulisboa.pt/gamecourse/>

system or any other LMS. The users were then asked to continue the experiment either to the new or the old version of the system (the users with odd id used first the old system then the new, and even ids the opposite). In each system, the users were given 5 minutes to navigate it and ask questions. After this time or the user approval to begin the test, we gave the user a list of tasks to perform:

1. Create a new course⁶.
2. Delete a course.
3. Make a user admin of the system.
4. On a course show the list of users that have the role Teacher.
5. On a course change the role of a user.
6. On a course add a user⁶.
7. On a course create a new role.
8. On a course activate a module.
9. On a course create a new Page.
10. On a course edit a page on the default Specializations and add new parts.
11. On a course edit a page on the default Specializations and change the content.
12. Access the documentation and search for a specific function.
13. Edit a user.⁷
14. Check the information about the user logged in⁷.
15. Verify if a module is installed on the system⁷.
16. Logout of the system and try to login with a Likedin account⁷.
17. On a course access the configuration page of a module and add a new item⁷.

These tasks were given randomly so there is no direct influence of the learning curve of the system. Check Appendix H for the detailed version of the tasks on each system, as they change due to system restrictions.

6.2 Results

From the list of tasks done, we divided the analysis into three phases. On the first one, we analyzed tasks 1 to 10 and 12. In the second phase, task 11. Finally, in the third phase, we analyzed tasks 13 to 17 that were only done on the system's new version.

We started by calculating the Success rate of each task, shown in Table 6.1. From these results, we learned that the success rate on the new version of the system is 100% in almost all tasks and higher than the old version, except for task 10, where the old version got 10% more success.

Then we calculated the mean, median, and standard deviation values of each task on both systems (Table 6.2 and 6.3). By looking at the mean value, we can see that the time spent on each task is lower on the new version for tasks 2, 5, 6, 8, and 12, but on the remaining tasks, the values are close. To have an

⁶This task is slightly different on the old system

⁷This task was only performed on the new version

Version	1	2	3	4	5	6	7	8	9	10	12
old	100%	85%	100%	100%	85%	95%	100%	100%	100%	95%	85%
new	100%	100%	100%	100%	100%	100%	100%	100%	100%	85%	90%

Table 6.1: Success rate of tasks 1 to 10 and 12 per version.

exact result that tells us if the change of the GUI had an impact on the time spent to complete the tasks, we tested the null hypothesis of "both versions have the same impact on the performance of each task". First, we did a Shapiro-Wilk Normality Test to verify if the data of each task follows a normal distribution on both versions. In Table 6.2, we have the results for the tasks which had normally distributed data, and in Table 6.3, the ones which did not. For the tasks on the first table, we did a Student's T-test to try to reject the null hypothesis, and on the second table, we did a Wilcoxon Signed-Rank Test.

	Version	1	2	3	4	5	7	9	10
Mean	old	00:23	01:15	00:17	00:17	01:03	00:44	00:24	02:48
	new	00:44	00:09	00:12	00:19	00:26	00:43	00:24	02:51
Median	old	00:21	00:35	00:15	00:13	00:50	00:34	00:21	02:35
	new	00:43	00:08	00:11	00:14	00:25	00:29	00:22	02:40
Std. Deviation	old	00:06	01:00	00:10	00:10	00:47	00:25	00:12	01:10
	new	00:11	00:03	00:06	00:20	00:07	00:31	00:08	01:04
Student's t-test	-	0,000	0,000	0,055	0,602	0,005	0,910	0,847	0,876
Null Hypothesis	-	R	R	NR	NR	R	NR	NR	NR
Fastest Version	-	Old	New	-	-	New	-	-	-

Table 6.2: Results for the tasks 1 to 5, 7, 9 and 10. (R - Rejected; NR - Not Rejected;)

	Version	6	8	12
Mean	old	01:46	00:25	02:34
	new	00:31	00:20	01:16
Median	old	01:37	00:22	02:29
	new	00:27	00:20	01:03
Std. Deviation	old	00:32	00:09	01:07
	new	00:11	00:05	00:49
Wilcoxon Signed-Rank	-	0,000	0,001	0,004
Null Hypothesis	-	Rejected	Rejected	Rejected
Fastest Version	-	New	New	New

Table 6.3: Results for the tasks 6, 8 and 12.

In order to reject the null hypothesis, both tests needed to return a significance $p < 0.05$. For tasks 2, 5, 6, 8, and 12, we were able to reject it and confidently say that the new version of the system allows a quicker performance on these tasks. On task 1, we were also able to reject the null hypothesis, however the mean value indicates the quicker system is the old one. This happens because the new version requires more input fields than the old one. Something similar happens on task 6, where the user had to place more information on the old version. So, we decided to test if "the impact of having more inputs is equal on both versions". We started by calculating the difference between versions on both tasks. difT1 represents the additional time spent to add more inputs on the new version, and difT6

on the old one (Table 6.4). Then we verified if the data followed a normal distribution, and as it did, we applied a Student's t-test. We obtained a $p = 0,000 < 0,05$ which rejects the null hypothesis, allowing us to conclude that the new version of the system has less impact on the time spent per task, when having more inputs to fill.

	Mean	Median	Std.	T-test
difT1	00:21	00:22	00:12	0,000
difT6	01:15	01:08	00:29	

Table 6.4: Results for the impact of having more inputs to fill.

Even though the View editor's look was changed, on task 11 the user needed to use the Expression Language, which was not altered. Thus, the results of the second version tested suffered a direct impact, especially if the user was able to complete the task on the first encounter. In table 6.5, we can see this exact influence as the user takes a lot less time performing the task the second time he/she encounters it, and the success rate also increases significantly in both cases. To test the null hypotheses of "both versions have the same impact on the performance of task 11" we considered use the first encounter of each version. However, this analysis was impossible to make. Due to the low success rate of this task, the amount of available data was not enough to make conclusions based on the first encounter. We then performed a Shapiro-Wilk Normality Test to verify if the data follows a normal distribution on both versions. As it does we applied a T-test, from which we got a $p = 0.318 > 0.05$. not rejecting our null hypothesis.

First Version Tested	Version	Success rate	Mean	Median	Std
old	old	30%	03:40	03:10	00:59
	new	80%	02:07	02:00	00:22
new	new	60%	03:35	04:06	01:17
	old	80%	01:17	00:50	00:43
-	old	55%	02:12	02:15	01:19
-	new	70%	03:21	03:46	01:19

Table 6.5: Results of task 11 separated by the first tested version.

For the last tasks, 13 to 17, we started by calculating each task's success rate (Table 6.6). All of them reported 100% success, except for 14, where one of the users did not understand what he/she was asked to do. Then we calculated the mean, median, std, and the 95% confidence interval values. Table 6.7 demonstrates that for task 13, the users took less than 40 seconds to complete it, for tasks 14 to 16 less than 20 second, and for task 17 less than 2 minutes, as it requires filling in data.

Version	13	14	15	16	17
new	100%	95%	100%	100%	100%

Table 6.6: Success rate of the tasks 13 to 17 on the new version.

	13	14	15	16	17
Mean	00:29	00:08	00:09	00:13	01:38
Median	00:28	00:05	00:09	00:13	01:38
Std. Deviation	00:10	00:09	00:03	00:03	00:33
95% Confidence Int.	[00:24-00:35]	[00:03-00:12]	[00:08-00:11]	[00:12-00:15]	[01:29-01:56]

Table 6.7: Results of tasks 13 to 17 on the new version.

6.2.1 NasaTLX

For the Nasa-TLX we used a variant called Raw TLX which includes only the sum of the results of the questions, instead of using weighted scales. We started by performing a Shapiro-Wilk Normality Test to verify if the data follows a normal distribution on both versions. The results show that there was indeed a normal distribution, which allow us to move to the Student's t-test. From this test we got a $p = 0.01 < 0.05$ rejecting the null hypothesis of "both systems having the same impact on the workload of the student" (Table 6.8). Therefore, we can conclude that the new version of the GameCourse has less impact on the workload of the user.

Version	N	Mean	STD	Std. Error Mean	Student's t-test
old	20	49,8000	26,82418	5,99807	0,010
new	20	29,8000	19,35159	4,32715	

Table 6.8: Results of the NasaTLX questionnaire.

6.2.2 System Usability Scale

From the data collected on the SUS questionnaire we started by calculating the SUS score:

- For each of the odd numbered questions, subtracted 1 from the score.
- For each of the even numbered questions, subtracted their value from 5.
- Added the previous results for the total score. Then multiplied it by 2.5.

We can immediately see by the final results (Table 6.9) for both versions that the new one has a much higher score with more the double the points. To prove the impact we first did a Shapiro-Wilk Normality Test, which indicated that the data did not follow a normal distribution on both versions. Then we performed a Wilcoxon Signed-Rank Test from which we got a $p = 0.00 < 0.05$. This rejected the null hypothesis of "both systems having the same level of usability" (Table 6.9) and proved that the new system has a greater level of Usability.

Version	N	Mean	STD	Confidence (95%)	Wilcoxon Signed-Rank
old	20	35,8	15,04598215	6,594071251	0,000
new	20	77,1	18,05100595	7,911056797	

Table 6.9: Results of the SUS questionnaire.

6.2.3 User Experience Questionnaire

To analyse the results from the UEQ we used an excel sheet provided by the platform in which we placed the results for both versions. It started by transforming the data so the values were between -3 and 3 instead of 1 and 7 and then calculating the mean for each scale (Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation and Novelty). From these values we can see that there is a clear distinction on the results of both versions (Table 6.10). We have the old version's mean on the negative side and the new version one on the positive side. In Figure 6.1 we can see that even though the mean for the old system is negative there are some results on the positive side on the Efficiency and Dependability scales. But the new version has always positive and higher results. To reveal the true impact of the change regarding User Experience we did a Shapiro-Wilk Normality Test that showed that not all the scales follow a normal distribution. Therefore we performed a Wilcoxon Signed-Rank Test that rejected the null hypothesis of "both systems provide the same level of User Experience" (Table 6.10) with all scales having a $p = 0.00 < 0.05$. This analysis proves that the new version has a better UX.

6.3 Discussion

The user tests allowed us to understand the impact of the new version of the system on the performance of the tasks and see where there is room for improvement. It showed us that the new tasks can be done quickly and without much effort. In addition: finding some information on the documentation takes half of the time it used to take; deleting a course takes one minute less to be completed; and that changing user's roles also takes half of the time it used to take.

Even though the visual change did not affect the time spent to complete some of the tasks, like adding new roles to a course, it improved the way a user sees and interacts with the system. All three questionnaires had a significantly better score on the new version of GameCourse, with almost half of the previous workload, twice the usability, and permanent positive results on UX.

The GUI of the system can still be improved, especially on the View editor, where the users had more difficulty getting through the tasks and where the success rate is below 50%. There were difficulties regarding the interaction with the Edit Layout option and the Edit Part, which created both stress and frustration on the user.

Scale	Version	N	Mean	STD	Wilcoxon Signed-Rank
Attractiveness	old	20	-1,38	0,96	0,000
	new	20	1,83	0,86	
Perspicuity	old	20	-1,24	1,08	0,000
	new	20	1,54	1,05	
Efficiency	old	20	-0,21	1,13	0,000
	new	20	2,05	0,59	
Dependability	old	20	-0,09	1,11	0,000
	new	20	1,93	0,65	
Stimulation	old	20	-0,70	1,20	0,000
	new	20	1,61	0,78	
Novelty	old	20	-1,25	1,17	0,000
	new	20	1,46	1,04	

Table 6.10: Analysis per system of the results of the UEQ questionnaire.

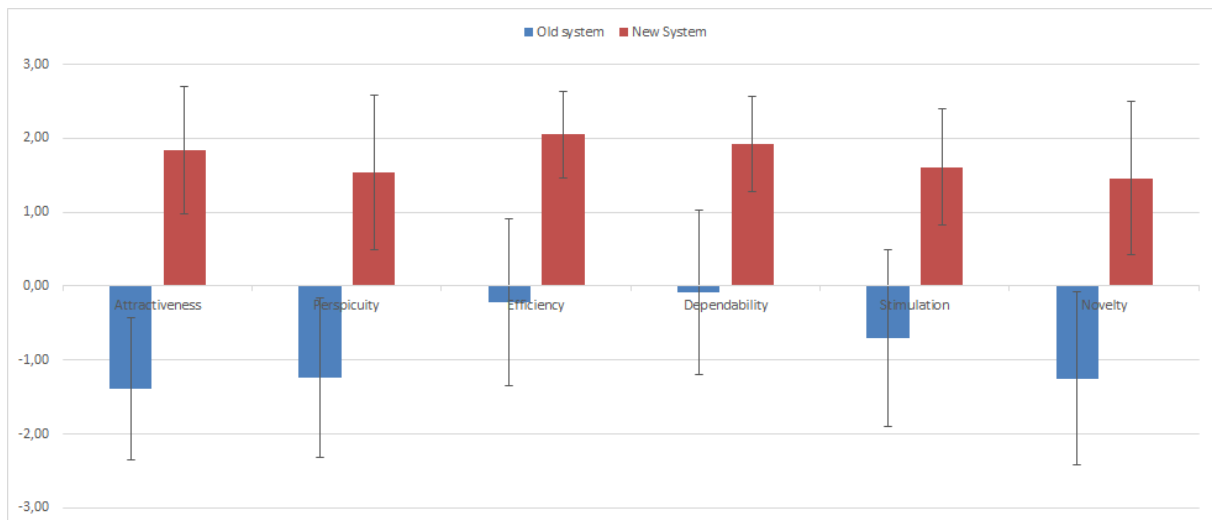


Figure 6.1: Mean value and Standard Deviation on each scale of the User Experience Questionnaire.

Chapter 7

Conclusion

For this thesis, we designed and implemented a new GUI for the GameCourse system that is clear, simple, minimalistic, and respects UI & UX principles. Besides restructuring the system so all the concepts are encapsulated on their correspondent pages, this thesis added new functionalities including: modules configuration pages; user configuration on the system level; search, filter and order capabilities on the listing pages; and undo/redo behavior on the roles management. These changes guaranty that all the use cases established for the system are available whilst improving the system's usability, making each task more accessible to the user.

The GameCourse now provides two versions of the **Courses page**: one where an admin user can find the courses registered on the system, manage their information, and access a course; other where the non-admin user can see and access all the courses in which he/she is enrolled. It also provides a new **Users page** on the system domain where all the user's information can be edited. This page on the course domain allows adding existing users and manage all course user's roles. There is also a search, filter, and order mechanism that allows quick access to a particular object, on these pages.

We have also implemented a new **state manager** that allows the user to undo and redo actions made while managing the list of roles, their hierarchy, and correspondent landing pages. The state manager was implemented independently, meaning that it can also be used on future pages.

On the **View editor**, the toolbar of each item will now appear fixed at the bottom of the page with bigger icons, after the correspondent part is selected by click. The pages triggered by each option on the toolbar were replaced by modals, that are used throughout the system to receive user's input. To take advantage of the visual aspect of this GameCourse's section and the user's ability to associate icons with tasks, some of the options on the modals were created with a visual selection instead of dropdown menus. To help understand this section and the expression language, the GameCourse now provides an always visible slider with shortcuts to the documentation pages. These pages were also changed to allow easy access to information.

We created a modular system that generates a **configuration page** for every module that requires it. This allowed us to remove the external scripts that fetch data for the module's configuration. This system allows the creation of configuration pages for both existing and future modules, which means that developers working on future modules only have to define the requirements in the module class instead of learning how to create a page on the system. This feature was essential since it guarantees consistency throughout the system whilst decreasing the developing effort.

The work of this thesis improved the system's usability, which now scores a 77 out of 100 compared with a previous 36. It also reduced the system's workload by almost half, leaving the users more satisfied and less stressed with the experience. Finally it brought all the user experience scale points to the positive side with mean values around 1 in a scale of -3 to 3.

7.1 Future Work

This system and its GUI have not yet reached its final state as some improvements can still be made. As proved by the final User tests, the View editor is still very painful to use and time-consuming. It would be important to implement the tutorial planned during the design phase, to add the aspect selection into the view editor, and perhaps find another solution for this part of the system.

Besides the View editor, the whole system needs additional code for a mobile version. Even though we had into consideration the resize of the window in multiple cases, making it flexible and dynamic, it is not enough to cope with very small widths of windows and small screens. This is especially needed because of the navbar and the sidebars.

Additionally, the "My Information" page would take advantage of editable inputs to let the user change his/her information, instead of having to request the change to a teacher.

With the current work, it is finally possible to implement the courses' themes to add a particular visual and take the gamification experience to a new level.

Bibliography

- [1] T. Oppenheimer, "The computer delusion," *The Atlantic Monthly*, vol. 280, no. 1, pp. 45–62, 1997.
- [2] L. Cuban, *Oversold and underused*. Harvard university press, 2009.
- [3] G. Morgan, "Improving Student Engagement: Use of the Interactive Whiteboard as an Instructional Tool to Improve Engagement and Behavior in the Junior High School Classroom," *Doctoral Dissertations and Projects*, oct 2008. [Online]. Available: <https://digitalcommons.liberty.edu/doctoral/121>
- [4] S. Addison, A. Wright, and R. Milner, "Using clickers to improve student engagement and performance in an introductory biochemistry class," *Biochemistry and Molecular Biology Education*, vol. 37, no. 2, pp. 84–91, mar 2009. [Online]. Available: <http://doi.wiley.com/10.1002/bmb.20264>
- [5] J. G. Ruiz, M. J. Mintzer, and R. M. Leipzig, "The impact of e-learning in medical education," *Academic medicine*, vol. 81, no. 3, pp. 207–212, 2006.
- [6] S. Kumar, A. K. Gankotiya, and K. Dutta, "A comparative study of moodle with other e-learning systems," in *2011 3rd International Conference on Electronics Computer Technology*. IEEE, apr 2011, pp. 414–418. [Online]. Available: <http://ieeexplore.ieee.org/document/5942032/>
- [7] "An Argument for Clarity: What are Learning Management Systems, What are They Not, and What Should They Become?" *TechTrends*, vol. 51, no. 2, pp. 28–34, mar 2007. [Online]. Available: <http://link.springer.com/10.1007/s11528-007-0023-y>
- [8] F. Grivokostopoulou, K. Kovas, and I. Perikos, "Examining the impact of a gamified entrepreneurship education framework in higher education," *Sustainability*, vol. 11, no. 20, p. 5623, 2019.
- [9] K. Erenli, "The impact of gamification-recommending education scenarios," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 8, no. 2013, pp. 15–21, 2013.
- [10] S. Y. Hock, R. Omar, and M. Mahmud, "Comparing the usability and users acceptance of open sources learning management system (lms)," *International Journal of Scientific and Research Publications*, vol. 5, no. 4, pp. 1–5, 2015.
- [11] A. H. Dourado, "Gamecoursenext," Master's thesis, Instituto Superior Técnico, Lisbon, 2019.

- [12] A. M. Baltazar, "Smartboards," Master's thesis, Instituto Superior Técnico, Lisbon, 2016.
- [13] N. Cavus and T. Zabadi, "A comparison of open source learning management systems," *Procedia-Social and Behavioral Sciences*, vol. 143, pp. 521–526, 2014.
- [14] T. Acosta and S. Luján-Mora, "Comparison from the levels of accessibility on lms platforms that supports the online learning system," in *8th annual International Conference on Education and New Learning Technologies*, 2016.
- [15] M. Croitoru and C.-N. Dinu, "A critical analysis of learning management systems in higher education," *Academy of Economic Studies. Economy Informatics*, vol. 16, no. 1, pp. 5–18, 2016.
- [16] M. J. Fonseca, P. Campos, and D. Gonçalves, *Introdução ao Design de Interfaces*, 10 2012.
- [17] "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, Dec 1990.
- [18] "Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts," International Organization for Standardization, Geneva, CH, Standard, Mar. 2018. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en>
- [19] A. Lodhi, "Usability heuristics as an assessment parameter: For performing usability testing," in *2010 2nd International Conference on Software Technology and Engineering*, vol. 2, Oct 2010, pp. V2–256–V2–259.
- [20] J. Nielsen, *Usability engineering*. Elsevier, 1994.
- [21] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1994, pp. 152–158.
- [22] M. Machado and E. Tao, "Blackboard vs. moodle: Comparing user experience of learning management systems," in *2007 37th annual frontiers in education conference-global engineering: Knowledge without borders, opportunities without passports*. IEEE, 2007, pp. S4J–7.
- [23] L. Martin, D. R. Martínez, O. Revilla, M. J. Aguilar, O. C. Santos, and J. G. Boticario, "Usability in e-learning platforms: heuristics comparison between moodle, sakai and dotlrn," in *Sixth International Conference on Community based environments. Guatemala*. Citeseer, 2008, pp. 12–16.
- [24] J. Thacker, M. Russell, and S. Brawley, "Learning management system comparative usability study," 2014.
- [25] D. S. S. Sahid, P. I. Santosa, R. Ferdiana, and E. N. Lukito, "Evaluation and measurement of learning management system based on user experience," in *2016 6th International Annual Engineering Seminar (InAES)*. IEEE, 2016, pp. 72–77.

[26] J. Nielsen, "How to conduct a heuristic evaluation," *Nielsen Norman Group*, vol. 1, pp. 1–8, 1995.

[27] W. L. Hürsch and C. V. Lopes, "Separation of concerns," 1995.

Appendix A

List of the 64 LMSs

- AdaptiveU
- Adobe Captive Prime
- Asentia LMS
- ATutor
- BlackBoard
- Bridge LMS
- Brightspace
- Callidus Cloud
- Canvas
- Chamilo
- Claroline
- Cornestone On Demand
- Docebo
- Dokeos
- Easy LMS
- eCoach
- edmodo
- EduBrite
- eFront
- Eliademy
- EthosCE
- Forma LMS
- Google classroom
- Goskills
- GosToTraining
- Grovo
- GrowthEngineering
- Halogen Software
- Ilias
- Inquisid
- iSpring Learn
- ItsLearning
- Kannu
- Kiwi LMS
- KnowMax
- LAMS
- LearningStone
- LearnUpon
- Lessonly
- LON-CAPA
- Looop
- Moodle
- Neo LMS
- OLAT
- Open edx
- OpenOLAT
- Opigno
- ProProfLMS
- Sakai
- SAP Litmos
- Schoology
- SkyPrep
- SWAD
- TalentLMS
- TalentCards
- Teachable
- Teamie
- Totara Learn
- Trainual
- Uduku
- Uqualio
- WebWork
- WizIQ
- 360 LEP

Appendix B

Websites with top X LMS lists

Capterra - top 20 most used:

<https://www.capterra.com/learning-management-system-software/#infographic>

Zapier top 8 best

<https://zapier.com/blog/best-lms/>

eLearning Industry top 20 best

<https://elearningindustry.com/the-20-best-learning-management-systems>

eLearning Industry top 10 Open source

<https://elearningindustry.com/top-open-source-learning-management-systems>

Finances Online top 10 best

<https://learning-management.financesonline.com/top-10-learning-management-software-solutions-for-your-company/>

Software Testing top 15 best

<https://www.softwaretestinghelp.com/learning-management-system/>

G2 top 10 and more

https://www.g2.com/categories/learning-management-system-lms?utf8=%E2%9C%93&order=g2_score

Software World top 10+ best

<https://www.softwareworld.co/top-learning-management-system-software/>

Appendix C

First User tests

C.1 Introduction

C.1.1 EN Version

Hello, my name is Patricia and I'm here to collect information about the usage of a system called Game-course, used on the MCP course on the Msc. of Information Systems and Computer Engineering. Beforehand I want to thank you for being here and help us analyse this system.

During this experiment I'll ask you to perform some tasks on the system and think-a-loud, saying whatever is on your mind while you use the system. This will help us discover potential confusing parts of the system. You are not being evaluated, the system is, so don't be afraid to speak.

What you say is important to us and because of that I'll be taking note. To make sure our notes correctly represent what you say, we would also like to take a sound recording. Of course, the recording is confidential and will not be shared around. If you have no objections, we'll proceed with the questions.

C.1.2 PT Version

Olá, o meu nome é Patrícia, desde já obrigada por te voluntariar a esta experiência! Estou aqui para recolher informações sobre o uso do sistema GameCourse, este sistema tem sido usado na cadeira de PCM do Mestrado de Engenharia Informática e de Computadores.

Durante a experiência vou-te dar tarefas para executar no programa e peço-te penses em voz alta, que digas aquilo que te vai na mente ao decorrer das tarefas. Ao fazeres isso vai-nos ajudar a encontrar potenciais problemas do sistema e partes nas quais ele seja confuso. Não tenhas medo de dizer o que te vai na cabeça, lembra-te que não és tu que estás a ser avaliado(a) mas sim o sistema.

Com esses teus pensamentos e ideias imediatas vamos poder produzir e criar um sistema melhor. Por isso mesmo enquanto fazes as tarefas vou tirar notas daquilo que fazes e dizes. Para ter a certeza de que a informação é bem anotada a tua voz e interação com o programa vão ser guardadas. Esta gravação é confidencial e não será usada para mais nada se não o propósito desta experiência. Tendo isto em conta, se não tiveres nenhuma oposição ou questão vamos começar.

C.2 List of Tasks

- Mostra a lista de cursos no sistema
- No curso PCM ativa o modulo Badges

- Cria um novo curso com o nome ABC
- Adiciona o seguinte aluno ao curso PCM:
 - Username: ist123456
 - Id: 123456
 - Name: João Silva
 - Email: joao@gmail.com
 - Campus: Alameda (A)
- No curso PCM adiciona ao user Carolina Xavier o role de Teacher e retira o role de Student , confirma na lista de Teachers.
- No curso PCM cria uma nova página de nome Perfil e com role single, acede à página e dá preview no final:
 - Adiciona um campo de imagem
 - Adiciona um campo de texto
 - Para o campo de imagem seleciona a imagem do viewer e torna o campo visível
 - Para o campo de texto seleciona o nome do view e torna o campo visível

C.3 Interview

C.3.1 EN Version

Task Execution

- Which is the task you find the most difficult to execute on the system? Why?
- While creating a new view, which are your biggest difficulties?
- Is it easy for you to find the configuration pages (about the course, views, pages, etc) on the system? If not, which are hard to find access to?
- Is there any particular action that is not clear on the system?

Expression language

- Was it obvious for you what which field meant, on the configuration page of an element?
- Was it easy for you to understand how to write the language and access the variables?
- Do you feel like the feedback near the field where you write the expression helped you?
- What would you change related to the expression language?

System Feedback

- When you find an error does the information provided by the system help you recover from it?
- Is the general feedback of the system useful? Is there any part of the system lacking feedback?
- Do you feel like you need any extra help feature in any page? If so, where?

Look & Feel

- Do you feel like there are more pages than needed? Or less? Which ones?
- What is your general opinion about the look of the system?
- Would you like to have any type of information on interaction shown in a different way? If so, explain what you would change.
- Is there any button that behaves unexpectedly? Is it doing more than expected or less?

C.3.2 PT Version

Execução de Tarefas

- Qual das tarefas te pareceu mais difícil de realizar? E porquê?
- Aquando da criação de uma nova view, qual foi a tua maior dificuldade?
- Sentiste que era fácil encontrar as páginas de configurações/definições? Se não, quais delas sentiste mais dificuldade em encontrar?
- Havia alguma ação em particular que não era clara ou óbvia no sistema? Se sim, o quê?

Expression language

- Foi obvio para ti o que significava cada secção da configuração de um elemento?
- Percebeste com facilidade como aceder ao valor pretendido através da expression language?
- Achas que o feedback sobre a expressão te ajudou?
- O que mudarias relacionado com a expression language?

Resposta do Sistema

- Quando encontras um erro sentes que a informação dada pelo sistema é suficiente para recuperar do mesmo?
- As respostas (feedback) do sistema são úteis? Há alguma parte do sistema onde gostarias de ter mais respostas do mesmo?
- Sentes que há necessidade de algum tipo de ajuda em alguma das páginas? Se sim, onde?

Aspeto do Sistema

- Sentes que existem mais páginas do que necessárias? Ou a menos? Quais?
- A nível visual qual foi a tua primeira impressão do sistema? E agora no final qual a tua opinião geral?
- Gostarias de ter algum tipo de informação ou interação feita de outra forma? Se sim, o que mudarias?
- Existe algum botão que se comporte de forma inesperada? Faz mais ou menos do que estavas à espera?

Appendix D

GameCourse Use Cases

The Use Cases are divided by permission level: Set A is only accessible by a GameCourse Admin, set B by a CourseUser with (Non-Edit) Professor Role, set C by a CourseUser with Student/Guest Role, and set D by external platforms:

- **A1** Create Course.
- **A2** Duplicate Course.
- **A3** Promote to Admin.
- **A4** Demote from Admin.
- **A5** Activate GameCourseUser.
- **A6** Deactivate GameCourseUser.
- **A7** Activate CourseUser.
- **A8** Deactivate CourseUser.
- **A9** Install Module.
- **A10** Uninstall Module.
- **A11** Delete Course.
- **A12** Restore to Snapshot.
- **B1** Edit Course Details.
- **B2** Export Course...
 - **B2.1** ... with Course Data.
 - **B2.2** ... without Course Data.
- **B3** Choose Theme.
- **B4** Import Theme.
- **B5** Change Profile.
- **B6** Enable Page, for instance ...
 - **B6.1** ... Award List.
 - **B6.2** ... Leaderboard.
 - **B6.3** ... Overview.
 - **B6.4** ... Profile.
 - **B6.5** ... Sideview.
- **B7** Disable Page, for instance ...
 - **B7.1** ... Award List.
 - **B7.2** ... Leaderboard.
 - **B7.3** ... Overview.
 - **B7.4** ... Profile.
 - **B7.5** ... Sideview.
- **B8** Enable Module, for instance ...
 - **B8.1** ... Badge.
 - **B8.2** ... Laboratory.
 - **B8.3** ... Project.
 - **B8.4** ... QRCode.
 - **B8.5** ... Quest.
 - **B8.6** ... Quiz.
 - **B8.7** ... Skill.
 - **B8.8** ... XP and Level.
- **B9** Disable Module, for instance ...
 - **B9.1** ... Badge.
 - **B9.2** ... Laboratory.
 - **B9.3** ... Project.
 - **B9.4** ... QRCode.
 - **B9.5** ... Quest.
 - **B9.6** ... Quiz.
 - **B9.7** ... Skill.
 - **B9.8** ... XP and Level.
- **B10** Create View...
 - **B10.1** ... From New.
 - **B10.2** ... From Template by Copy.
 - **B10.3** ... From Template by Reference.
- **B11** Edit View.
- **B12** Delete View.
- **B13** Create Aspect.
- **B14** Edit Aspect.
- **B15** Delete Aspect.

- **B16** Configure Module, for instance ...
 - **B16.1** ... Badge.
 - **B16.2** ... Laboratory.
 - **B16.3** ... Project.
 - **B16.4** ... QRCode.
 - **B16.5** ... Quest.
 - **B16.6** ... Quiz.
 - **B16.7** ... Skill.
 - **B16.8** ... XP and Level.
- **B17** Add Role.
- **B18** Remove Role.
- **B19** Import Students.
- **B20** Create User.
- **B21** Update Course Data.
- **B22** Grade Module, for instance ...
 - **B22.1** ... Badge.
 - **B22.2** ... Laboratory.
 - **B22.3** ... Project.
 - **B22.4** ... QRCode.
 - **B22.5** ... Quest.
 - **B22.6** ... Quiz.
 - **B22.7** ... Skill.
 - **B22.8** ... XP and Level.
- **B23** Create Page.
- **B24** Delete Page.
- **B25** Set Template as Global.
- **B26** Add External Data Source.
- **B27** Import Module.
- **B28** Export Module.
- **B29** Open developer terminal.
- **C1** Log in, for instance via ...
 - **C1.1** ... Facebook.
 - **C1.2** ... Fénix.
 - **C1.3** ... Google.
- **C2** Notify.
- **C3** Disable Notification.
- **C4** Access Page, for instance ...
 - **C4.1** ... Award List.
 - **C4.2** ... Leaderboard.
 - **C4.3** ... Overview.
 - **C4.4** ... Profile.
 - **C4.5** ... Sideview.
- **D1** Collect Course Data.

Heuristic Evaluation Results

Broken Heuristic	Description	Severity
4,8	Action to add a role to a user does not look like a button	4
1,2,5	Verbal time for "enable" label is wrong	4
3	on the content option "text/exp" is not exclusive	4
4	Settings inside and out of the Course seem the same.	3
2,4	Label "Move" is not self explanatory	3
4,5,6,9	current icon on the role removing part is not clear	3
5,6	Save action should be the spotlight, is the most important task	3
2	Adding different views of an element depending on a role takes a lot of steps and is confusing	3
2	Courses is missing an attribute - "visible"	3
7	edit and view of user look the same, no need for both	3
5,10	Tutorial should include more information about visible on condition and events, is not clear	3
1	Generate cookies to save login information - show to the user the account with which they are going to login	3
6,8	Colours are not scalable as a label	3
2	Courses attributes (labels) like Year and Active are not clear	3
2,6,8	Element editor is crazy - diabolic	3
6	Let user know what's inside of the Settings section	2
2,4	Double functionality (active/not active) of the buttons Preview, Edit and Tutorial is not clear	2
6	Allow the user to fill the inputs while the tutorial is playing	2
1	Add a confirmation/success message after adding a new object	2
10,6	Import and Export buttons do not have a label	2
9	There is nothing saying why some roles can not be moved or deleted	2
4	Have the help buttons always on the same side	2
2	Label "Exp" is not clear	2
5	Removing a role from a user should have a confirmation message	2
3,7	Add search, filter and order option for modules and pages	2
8,10	The spotlight on the login page is not well distributed	2
4	Order the options on the documentation (alphabetic or logic) and filter and order	2
7	shortcut selection by role	2
2	Users missing an attribute - "nickname"	2
5,9	moving table columns is confusing and misleading	2
3	Variables name should be editable as well	2

7	Allow drag & drop from the help - variables to use on the inputs	2
4	Reorganize Edit Element modal - label should be on top and visibility near events	2
10	%user and %viewer while creating a new view should be explained	2
8	Actions only visible on hover and not always	2
4	Related labels should say the same "active" != "activate course"	2
2	The main page section when selected "default" is not clear what it means	2
2	Action of adding a new role is not consistent with the other pages	2
2	Na Enabled Template is not clear on what it means	2
1	Organization of pages is not clear	2
8	Place settings options on a 2 ^o level navbar	1
6,8	Reorganize color input while creating new course	1
7	Only the name of the course is clickable	1
7	You can not create or alter the roles while editing a user	1
4	not all of the roles have a delete icon	1
6	Is not clear what the page of the Modules allows us to do	1
4	Change order of inputs to select roles of the user	1
4,6	is not clear that some sections of help open	1
3,4,6	Have exit option on the top bar of the tutorial	1
5,8	Diagonal arrows are confusing	1
8	for bigger cards color of active/inactive is not visible	1
4	Use "add" instead of "continue" when adding a new element to the view	1
4,8	IST logo is not of the same style of the others	1
2	Add a game card for each user	1
3,7	Allow drag & drop between templates and pages, to generate a new page from a template	1
8	All the prototype elements are too big	1

Documentation section about Modules configuration

F.0.1 Module Configuration

You can setup a configuration page for your module to receive information from admin users. You can have a section for general type of inputs, a more personalized section and a place for listing items of the module and manage them (add, edit, delete). Declare your configuration page

First thing you need to do is to declare that your module will have a configuration page, that is as simple as including this function on your php file:

```
1 public function is_configurable(){
2     return true;
3 }
```

Then you'll need to inform the system that there is a new url being used for that configuration page. If you don't already have one, create a new .js file inside a folder called js on your directory. There include the following declaration: (Remember to always use your module id as reference)

```
1 app.stateProvider.state('course.settings.empty', {
2     url: '/empty',
3     views : {
4         'tabContent': {
5             controller: 'ConfigurationController'
6         }
7     },
8     params: {
9         'module': 'empty'
10    }
11 });
```

F.0.1.A General Inputs Section

On this section you can define which inputs you want on the configuration page. For that we use three functions on the Module file: `has_general_inputs()` which is already set to return false if you don't need this section, `get_general_inputs ($courseId)` so you can inform the controller (front-end) of

which inputs you need and `save_general_inputs($generalInputs,$courseId)` so you can receive the information back when the user saves.

To use this section first declare public function `has_general_inputs ()`{ return true; } Then specify which inputs you want through get function. You'll need to define the 'name' you want to show for the input, it's 'id' for internal use and reference, a 'type' which can be any of the following: text, date, on_off button, select, color, number or paragraph. 'options' if you choose the select type and 'current_val' to show already defined information. Below there is an example of the possible types of inputs.

```
1 public function get_general_inputs ($courseId){
2     $input1 = array('name' => "input 1", 'id'=> 'input1', 'type' => "text",
3     'options' => "", 'current_val' => "things");
4     $input2 = array('name' => "input 2", 'id' => 'input2', 'type' => "date",
5     'options' => "", 'current_val' => "");
6     $input3 = array('name' => "input 3", 'id' => 'input3',
7     'type' => "on_off button", 'options' => '', 'current_val' => true);
8     $input4 = array('name' => "input 4", 'id' => 'input4', 'type' => "select",
9     'options' => ["OpA","OpB","OpC"], 'current_val' => "");
10    $input5 = array('name' => "input 5", 'id' => 'input5', 'type' => "color",
11    'options' => "", 'current_val' => "#121212");
12    $input7 = array('name' => "input 7", 'id' => 'input7', 'type' => "number",
13    'options' => "", 'current_val' => "");
14    $input8 = array('name' => "input 8", 'id' => 'input8',
15    'type' => "paragraph", 'options' => "", 'current_val' => "my text here");
16    return [$input1, $input2, $input3, $input4, $input5, $input7, $input8];
17 }
```

This section has a save button associated, when the user clicks on it, a save function will be called. There you'll receive the course id and general inputs information with the pairs id-input so you can save the information as you need.

```
1 public function save_general_inputs($generalInputs,$courseId){
2     $input = $generalInputs["input_id"];
3     $this->saveInput($input, $courseId);
4 }
```

F.0.1.B Listing Items Section

On this section you can list, add, edit and delete your module's items. For that to happen we use 3 functions: `has_listing_items ()` that by default is set to false, `get_listing_items ($courseId)` so

you can define the table and items structure, and `save_listing_item` (`$actiontype`, `$listingItem`, `$courseId`) so you can save the information gather from the actions new, edit and delete. Don't worry we show a confirmation message before submitting any delete action.

To use this section first declare public function `has_listing_items(){ return true; }` Then you must gather your table and items information: choose which item's attributes you want to show on the table (in case they are a lot, choose a set of them) save the id of those attributes and the name you want to give to that column on separate arrays but in the same order (`displayAtributes` and `header` correspondently). On another array place all your items information, including each item id. Then similar to the general inputs create an array with the information of each attribute and its type so we are able to create an add modal so the user can create a new item or edit an existing one. In this section the types available are: text, select, number, date and on_off button. To finish you'll need to define what it's name of your item and the items list. Return all of the above information on our get function.

```
1 public function get_listing_items ($courseId){
2     $header = ['Name', 'Description', 'XP', 'Is Point'] ;
3     $displayAtributes = ['name', 'description', 'xp', 'isPoint'];
4     $items = getItems();
5     $allAtributes = [
6         array('name' => "Name", 'id'=> 'name', 'type' => "text", 'options' => ""),
7         array('name' => "Description", 'id'=> 'description', 'type' => "text",
8             'options' => ""),
9         array('name' => "XP", 'id'=> 'xp', 'type' => "number", 'options' => ""),
10        array('name' => "Levels", 'id'=> 'levels', 'type' => "number",
11            'options' => ""),
12        array('name' => "Is Point", 'id'=> 'isPoint', 'type' => "on_off button",
13            'options' => "")
14    ];
15    return array( 'listName' => 'Badges', 'itemName' => 'Badge', 'header' => $header,
16        'displayAtributes' => $displayAtributes, 'items' => $items,
17        'allAtributes' => $allAtributes);
18 }
```

In each action, add, edit or delete, the configuration API will be called and information will be delivered at the save function. There you'll have to analyse the 3 possible actions: On the add case you'll receive the pairs `atribute_id-value` for each of the attribute you specified above, on the var `$listingItem`. In case of edit you'll receive the same information as in the add case plus the id of the item selected to edit. On the delete case on the `$listingItem` you'll receive only the pair with the id value of the item to be deleted. Be aware that on_off fields with boolean values, while the value true comes as 1, false comes empty instead of 0.

```

1 public function save_listing_item ($actiontype, $listingItem, $courseId){
2     if($actiontype == 'new'){
3         newItem($listingItem);
4     }
5     elseif ($actiontype == 'edit'){
6         editItem($listingItem);
7
8     }elseif($actiontype == 'delete'){
9         deleteItem($listingItem['id']);
10    }
11 }

```

F.0.1.C Personalized Section

If you need something more complex you can create your own section, build the front-end part and the API and resources needed. You'll just have to declare public function `has_personalized_config ()` { return true;} and tell which function the controller needs to call in order to build that part of the page.

```

1 public function get_personalized_function(){
2     return "personalizedConfig";
3 }

```

The controller will call that function and send as arguments the `$scope`, `$element` where you can attach your HTML structure, `$smartboards` in case you need it and the `$compile` so you can relate `$scope` information to HTML structure.

```

1 function personalizedConfig($scope, $element, $smartboards, $compile){
2     ..
3 }

```

List of tasks available on the LFP

- Activate the user João Costa.
- Add a new course called "Introdução à Engenharia Informática", short: IEI, year: 2020, color: 993300 and make it not active.
- Delete the course "Nome" with 56 students.
- Verify if the module Quest is installed on the system.
- On the course PCM add the role of "Teacher" to the user Patricia Costa and remove the role of "Student" .
- On the course PCM add a new role descendent from "Student" called "Attendee".
- On the course PCM enable the module QR.
- On the course PCM add a new page called "My Badges", role type: single, enable and edit it:
 - Add a block with the header saying "My badges"
 - Inside the block add an image
 - Make this image show every badge's images of the course
 - Save and access the page
- On the course PCM edit the page "Perfil":
 - Add a new version for the user: "Teacher";
 - Exchange the first two columns;
 - Preview both versions of the page.

List of tasks for the final User Tests

H.1 Old Version

1. Create a course:
 - Name: Visualização de Informação
2. Delete the course "Inteligência Artificial".
3. Make the user John Doe admin of the system.
4. On the course PCM show the list of users that have the role Teacher.
5. On the course PCM add the role of Teacher to the user Hermione Granger and remove the role of Student.
6. Add the following user as student to the course PCM:
 - Name: Ron Weasley
 - Id: 18463
 - Email: ron@gmail.com
 - Campus: T
 - Username: Ronnie
7. On the course PCM create a new role named Assistant descendant of the role Teacher.
8. On the course AD activate the module QR.
9. On the PCM course create a new Page:
 - Name: My Page
 - Role type: single
10. On the PCM course edit the page Example on the default Specializations:
 - Add a new block with a header saying "Example page"
 - Inside add a Table part with 2 columns and headers.
11. On the PCM course edit the Page My students on the default Specializations.
 - Currently this page shows a block per student with his name on the header and inside his email
 - Change this page as each block should contain the student picture
 - Explore other view parts on this page and the documentation to be able to finish the task
12. Access the documentation while on the page My Students on the default Specializations of the PCM course, how can I select all the courses on the system with the expression language?.

H.2 New Version

1. Create a course:
 - Name: Visualização de Informação
 - Short: VI
 - Year: 2020-2021
 - Color: 4B9345 (green)
 - Active
2. Delete the course "Inteligência Artificial".
3. Make the user John Doe admin of the system.
4. On the course PCM show the list of users that have the role Teacher.
5. On the course PCM add the role of Teacher to the user Hermione Granger and remove the role of Student.
6. On the course PCM add the user Ron Weasley to the course with the role of Student (the user already exists)
7. On the course PCM create a new role named Assistant descendant of the role Teacher.
8. On the course AD activate the module QR.
9. On the PCM course create a new Page:
 - Name: My Page
 - Role type: single
10. On the PCM course edit the page Example on the default Specializations:
 - Add a new block with a header saying "Example page"
 - Inside add a Table part with 2 columns and headers.
11. On the PCM course edit the Page My students on the default Specializations.
 - Currently this page shows a block per student with his name on the header and inside his email
 - Change this page as each block should contain the student picture
 - Explore other view parts on this page and the documentation to be able to finish the task
12. Access the documentation while on the page My Students on the default Specializations of the PCM course, how can I select all the courses on the system with the expression language?.
13. Edit the user John Snow and update his information:
 - Name: Aegon Targaryen
 - Nickname: Snow
14. Check the information about the user logged in.
15. Verify if the module Badges is installed on the system.
16. Logout of the system and try to login with a LinkedIn account.
17. On the course PCM access to the configuration page of Badges and add a new badge:
 - Name: Positive
 - Description: get positive on a test
 - Level 1: positive at first test
 - Level 2: positive at second test
 - XP 1: 1000
 - XP 2: 1000
 - isPoint : true
 - Count1: 1
 - Count2: 2

Appendix I

Schema of GameCourse database

I.1 Old Version

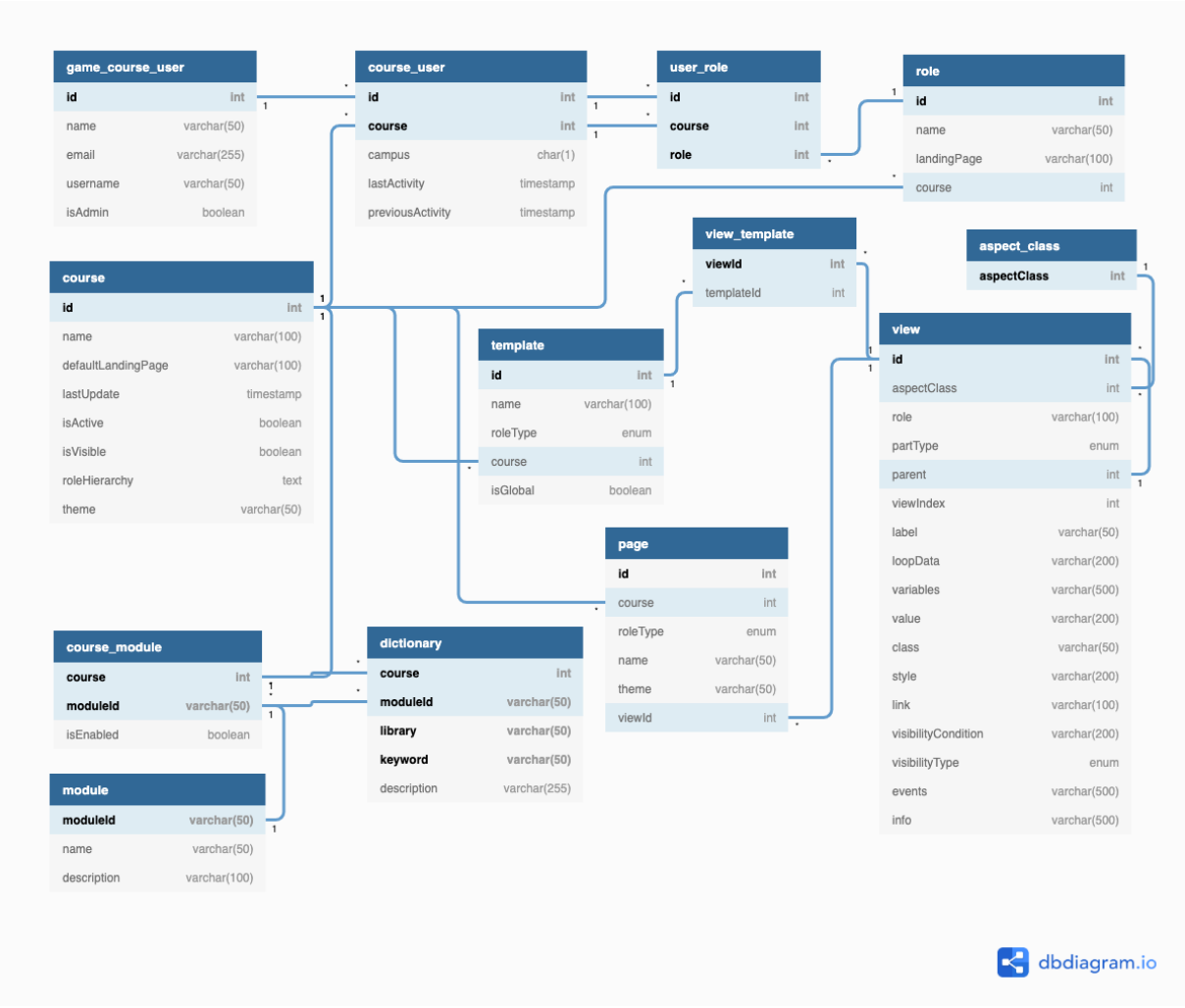


Figure I.1: Complete Schema of GameCourse database on the initial GameCourse.

I.2 New Version

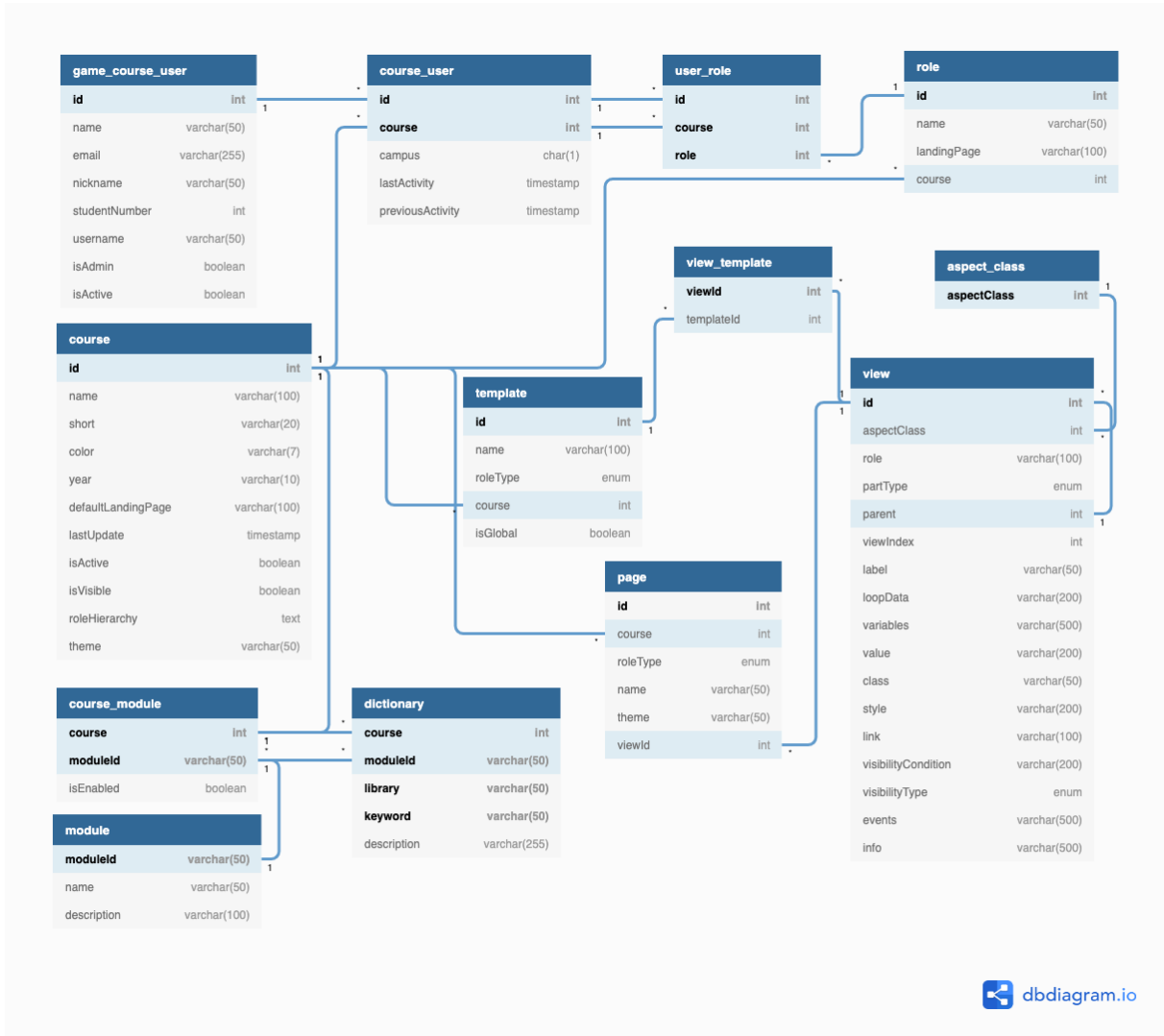


Figure I.2: Complete Schema of GameCourse database on the final version of GameCourse.