TÉCNICO LISBOA

GameCourseNext

Alice Horta Dourado

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Daniel Jorge Viegas Gonçalves
Prof. Joaquim Armando Pires Jorge

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. Rui Filipe Fernandes Prada

November 2019

Acknowledgments

I would like to thank my family for their unconditional support and encouragement throughout all these years.

I would also like to express my gratitude towards Professor Daniel Gonçalves, for supervising my thesis and for all the meetings and discussions we had which helped this project reach its completion. Another person who provided great help in the project of this thesis was Tomás Alves, who I thank for all the quick responses to questions or requests of revisions.

This work was partly supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) grants GameCourse PTDC/CCI-CIF/30754/2017 and UID/CEC/50021/2019, and by Instituto Superior Tecnico (IST) under grant BL269/2018 in the GameCourse project (1018P0543701).

I also want to thank all my friends from Instituto Superior Técnico, for helping to keep me motivated and relaxed. And to my friends from home, who have not been seeing me as much but are always there for me, I want to thank them for their friendship.

To each and every one of you, thank you.

Abstract

Gamification is the use of game design elements in non-game contexts. In recent years, this technique has been applied in education as a way to increase motivation for students to perform coursework. In particular, the Multimedia Content Production (MCP) course at IST was gamified using a web application that consists of loosely connected parts which form a non-cohesive system with scalability and usability issues. We developed a new gamification solution which is configurable, flexible and scalable, allowing it to be applied to courses other than MCP. Additionally, we implemented new features to support novice users by allowing them to directly interact with the platform without the need to have a deep knowledge of the application to produce code as in previous versions. Benchmark tests addressing loading speed showed positive results for an intermediate version of the platform. There were also user tests which showed significant improvements over the old system in terms of time to execute a set of relevant tasks. These tests also showed significantly better scores in the System Usability Scale and NASA Task Load Index questionnaires. Finally, a case study comparing both versions showed that the new solution is preferable to the old one in terms of usability and functionalities available.

Keywords

Gamification, Motivation, Education, View, Web Application

Resumo

Gamificação é o uso de elementos de *design* de jogos em contextos fora de jogo. Nos últimos anos, esta técnica tem sido aplicada a educação como forma de aumentar motivação de alunos para realizarem o seu trabalho escolar. A disciplina Produção de Conteúdos Multimédia (PCM) no IST foi gamificada usando uma aplicação web consistida por partes conectadas entre si que formam um sistema não coeso com problemas de escalabilidade e usabilidade. Nós desenvolvemos uma nova solução de gamificação que é configurável, flexível e escalável, podendo ser aplicada a várias disciplinas para além de PCM. Além disso, nós implementámos novas funcionalidades para apoiarem utilizadores novatos permitindo-lhes interagirem diretamente com plataforma sem ser necessário terem um grande conhecimento da aplicação para produzir código como nas versões anteriores. Testes de *benchmark*, para medir a sua velocidade de carregamento mostraram resultados positivos para uma versão intermédia da plataforma. Também foram realizados testes de utilizadores que mostraram melhorias significativas em relação ao sistema antigo em termos de tempo para executar um conjunto de tarefas relevantes. Estes testes também mostraram resultados significativamente melhores nos questionários de escala de usabilidade do sistema e de carga de tarefa da NASA. Finalmente, um caso de estudo a comparar ambas as versões, mostrou que a nova solução é preferível à antiga no que toca a usabilidade e funcionalidades disponíveis.

Palavras Chave

Gamificação, Motivação, Educação, Vista, Aplicação *web*

Contents

List of Figures

List of Tables

Acronyms

IST	Instituto Superior Tecnico
XP	Experience Points
MCP	Multimedia Content Production
SDT	Self-Determination Theory
MUD	Multi-User Dungeon
RPG	Role-Playing Game
VLE	Virtual Learning Environment
LMS	Learning Management System
ILS	Index of Learning Styles
VM	Virtual Machine
SUS	System Usability Scale
NASA-TLX	NASA Task Load Index
PCM	Produção de Conteúdos Multimédia
JSON	JavaScript Object Notation
FCT	Fundação para a Ciência e a Tecnologia

1

Introduction

Contents

Education is an important part of life that allows people to gain the skills and knowledge necessary to succeed in their professional careers. However, students can often feel unmotivated to study or disinterested in their classes which can cause academic failure. Therefore, it is useful to innovate the learning process in order to make it more appealing.

Nowadays there are many Virtual Learning Environments (VLEs) and multiple other web-based solutions to improve and modernize education. Neverthelesss, this can still not be appealing enough for those people who get easily demotivated, therefore a possible approach would be to turn the learning process into something similar to a game.

Gamification is "the use of game design elements in non-game contexts" [1]. It can be applied in many different areas [2] including education. There is a variety of elements that can be used for gamification. The most common are points, badges, and leaderboards. When designing a gamification experience it is important to analyze which elements are the best fit for the context; for example, leaderboards can be helpful because of the feedback that they give to users, however, they increase competition which may be considered a unfavorable. Thus, gamification should be carefully planned, especially if it is being applied to a classroom setting because it will affect the learning process of students. When gamification is successful it can make the process of acquiring knowledge more interesting to students and motivate them to study [3–5].

If an educator wants to provide a gamified experience, they could use an already existing VLE such as Moodle. Yet, if they want something tailored specifically to their needs, they have to create their own system with the exact customization that is needed to fit their requirements.

There is a gamified course at Instituto Superior Técnico called Multimedia Content Production (MCP) that has had success in engaging students. This course has a webpage called SmartBoards, which displays a leaderboard and students' profile pages. However, the overall system being used, called GameCourse, which consists of many parts including Moodle and SmartBoards, has integration and scalability problems and can be difficult to configure and update. A good way to improve this situation would be to have a configurable and scalable platform that could be adapted to a multitude of courses and their respective needs.

The goal of this thesis is to **create a scalable, flexible and stable gamification solution for education to replace the existent system of the MCP course.** This new system should have better response times, its configuration should be extendable, adjustable and effective, allowing for the gamification elements to be edited using the interface instead of having to modify their information on text files. Furthermore, the system should be scalable, allowing the possibility of being used for multiple courses simultaneously, gamified in different ways.

To achieve this, we designed and implemented a new solution. It was developed by extending the old gamification platform of the MCP course. Its database was completely restructured, and it got the

necessary improvements and added functionalities. Additionally, some of the disparate parts of the system were consolidated. While the solution was being developed, we also assured the maintenance of multiple parts of the system that was being used in MCP.

In order to evaluate the solution, we performed benchmark tests which measured load times of pages, and user tests to compare the old and new versions of the system in terms of usability and overall quality.

1.1 Document Outline

This document is divided into six chapters. Following the introduction, we have Chapter 2 with Related Work where we define gamification and cover related topics such as intrinsic motivation, game elements, gamification in education and gamification in the MCP course. Then, in Chapter 3 we go into detail about the initial state of the GameCourse system and its main component, SmartBoards. Afterwards, we have Chapter 4, where the implementation of this project is described as well as the functionalities of the developed system. Chapter 5 covers the evaluation performed on the system, including a benchmark test and user tests. Finally, we have Chapter 6 with the conclusion and future work.

2

Related Work

Contents

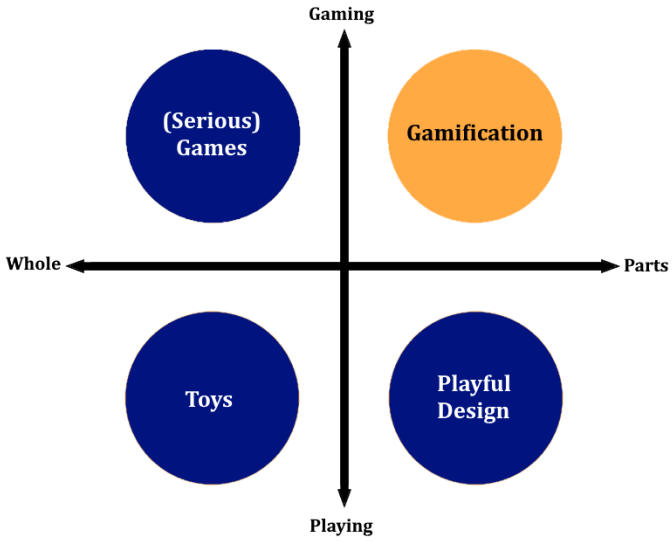


Figure 2.1: Gamification in relation to Games, Toys, and Playful Design. Adapted from Deterding *et al.* [1]

2.1 Gamification

Gamification is a strategy often implemented to improve user engagement and motivation in a way that influences them to adopt desired behaviors, it can be applied to various contexts such as health, education, and productivity. This concept has gained popularity over the years and is now used in many applications, but it needs to be thoughtfully designed in order to provide a beneficial experience to users.

Deterding *et al.* defined gamification as “the use of game design elements in non-game contexts” [1]. They also mention two dichotomies: gaming as opposed to playing and using a whole artifact as opposed to parts of it. These help differentiate gamification from the concepts of serious games, toys, and playful design (Fig. 2.1). This illustrates the idea that toys are just for unstructured fun activities, while gamification is something that has rules like games do, which differentiates it from playful design, and it is not a whole game itself like serious games, gamification just uses elements of games.

There was a research reviewing gamification studies, that aimed to answer the question ”Does Gamification Work?” [2]. The studies had various contexts, with the majority being education, and they varied in terms of game elements used, the most commonly tested were points, leaderboards and badges. The results showed that the gamification experiences had mostly positive effects, but it is possible that these could be caused by a novelty effect. All of the studies reported positive experiences from gamification but there often were a few people that disliked some of its aspects.

2.2 Intrinsic and Extrinsic Motivation

When studying gamification, it is important to understand what motivates people to perform tasks, and what type of motivation can be beneficial or harmful.

Intrinsic motivation exists when someone does an activity for its inherent satisfactions rather than some external consequence [6]. Which means that a person needs to have interest in an activity to create intrinsic motivation, it could be fun, challenging, enjoyable, or have other characteristics that incites the will to do it.

Extrinsic motivation refers to doing an activity in order to attain some separable outcome [6]. This can vary in degree of autonomy, for example a student performing a task because they fear parental sanctions and want to avoid punishment is extrinsically motivated, just like if the student does the task in order to have a good grade. In both cases the student is motivated to execute the task by external factors, however in the second one the student has a higher level of autonomy because they have control over the choice to do it which can lead to better engagement.

Ryan and Deci explained the Self-Determination Theory (SDT) about human motivation and personality [7]. According to this theory there are three innate psychological needs that are the basis for self-motivation and personal well-being:

- *Competence* is "the ability to do something successfully or efficiently"[1]. When someone receives feedback that leads towards feelings of competence for a task, it can enhance their motivation to perform that task.
- *Relatedness* is about feeling connected to others. If there is a sense of security and relatedness, intrinsic motivation is more likely to flourish.
- *Autonomy* is "self-directing freedom and especially moral independence"[2]. Feelings of competence should be accompanied by a sense of autonomy in order to cause intrinsic motivation, because it is important for people to know when they cause their accomplishments.

2.3 Player Types

When designing games or gameful experiences it can be useful to understand player types in order to create something that is suited for each user's personality.

In 1996, Bartle identified four types of player present in Multi-User Dungeons (MUDs) [8]: *Achievers* are players that care mostly about leveling up and gathering points. *Explorers* enjoy discovering un-

[1] https://en.oxforddictionaries.com/definition/competence
[2] https://www.merriam-webster.com/dictionary/autonomy

known places in the game, looking for interesting features and figuring out how things work. *Socializers* may be more interested in interacting with other players than on the game itself, which they use as a way to develop inter-personal relationships. *Killers* are the players that enjoy causing havoc and killing the characters of the other players. These types can also be characterized by two dichotomies of player interest, *Achievers* and *Explorers* care about the World, while the others care about the Players; *Socializers* and *Explorers* like Interacting, while the rest prefer Acting.

Another model created more recently is the BrainHex [9] which presents seven different architypes based on player traits, which are all associated with specific neurotransmitters and brain regions based on neurobiological research.

The *Seeker* is interested and curious about the world of the game, this type relates to the hippocampus which the part of the brain that processes memory and sensory information. *Survivors* enjoy the thrill of intense and frightening experiences that allow them to feel an adrenaline rush. The *Daredevil* likes taking risks such as being in high speed while still in control and navigating dangerous platforms. This type and the *Survivor* both relate to the amygdala. The *Mastermind* is someone who enjoys solving puzzles and devising strategies to solve problems, focusing on making efficient choices. They rely on the decision center of the brain. The *Conqueror* type describes players that want to struggle until they achieve victory, facing adversity that adds difficulty to their tasks, but they can eventually defeat. These players are often motivated by anger which is controlled by the hypothalamus. *Socializers* are players that like talking to other people, often being helpful and trusting, this type is associated with oxytocin, a neurotransmitter connected with trust. *Achievers* are goal-oriented players, motivated by the will to complete objectives. They feel satisfaction when attaining goals related to the release of dopamine into the pleasure center of the brain.

The User Types Hexad is another approach to defining player types, this one focuses specifically on gamification [10, 11]. This model defines six player types but accepts that people will probably have characteristics of multiple types. The first four types are intrinsically motivated, and some of these are related with specific SDT needs.

Socializers are motivated by relatedness, they want to interact with other users and create social connections. These people appreciate the elements of teams and social networks. *Free Spirits* enjoy exploring and being creative, they want to have freedom from external control and are motivated by autonomy. They like Easter eggs and exploratory tasks. *Achievers* want to complete every challenge, learn skills and improve themselves while trying to be better than other players, they are motivated by competence. For them the best game elements are challenges, quest and levels. *Philanthropists* are motivated by purpose, they are altruistic and want to help others by enriching their lives in some way.

They enjoy collecting, trading and gifting. *Players* are motivated by extrinsic rewards they may have behaviors similar to the previously mentioned types but the reason for their actions is the expectation of gaining some prize. Their favorite elements are points, achievements and leaderboards. *Disruptors* want to cause some kind of change. They may be interested in impacting the system or other players in harmful ways that can worsen the experience for others, or they might try to influence others in attempts to improve the experience. They like voting mechanisms and development tools.

2.4 Game Elements

Game elements can be divided in three main types [12] depending on their level of abstraction:

Game Dynamics are at the highest level of abstraction and they encompass general aspects of games such as constraints, emotions, narrative, progression, and relationships.

Then we have *Mechanics* which are "processes that drives the action forward" and they help achieving the dynamics previously mentioned. The Mechanics listed are: challenges, change, competition, cooperation, feedback, resource acquisition, rewards, transactions, turns, and win states.

The final type of element is the *Components* which are more specific aspects of games. These can be considered the most important components: achievements, avatars, badges, boss fights, collections, combat, content unlocking, gifting, leaderboards, levels, points, quests, social graphs, teams, and virtual goods. While these components may be important that does not mean that all of them are needed for a gamification experience, since they should be carefully chosen by the designers according to what best fits the context.

Different elements can help supporting the needs of SDT which may increase motivation. Autonomy can be boosted by the use of profiles, avatars, and elements that provide the ability to make choices, allowing the user to personalize their experience. Competence is supported by positive feedback, challenges, points, levels and leaderboards. Relation can increase by using social elements such as cooperation, chats and groups [13].

In a study made by Mekler et al. they evaluated points, levels and leaderboards, to see if these game elements influenced the motivation of users [14]. It showed that users increased their performance with these elements. The most significant change was with leaderboards and levels, which may be due to the existence of goals for participants to aspire to achieve. According to their findings even though these elements influenced the performance, the authors do not believe it affected intrinsic motivation in negative or positive ways. Therefore the addition of these elements can boost user performance, but it is not sufficient to make something more engaging, and when deciding if we should use these elements,

it is highly important to take into account the situational factors to prevent damaging intrinsic motivation.

2.5 Gamification Examples

As stated before, gamification can be applied to many areas. This chapter will present some of these, with an example of a gamification system for each one. The topics approached here are health and fitness, marketing, and programming. The first two were chosen due to being areas where gamification is widely used, and the latter was chosen because it is related to the area of Computer Science.

2.5.1 Health and Fitness: Fitocracy

Fitocracy[3] is a fitness website that allows users to track their exercises. They use many game elements to keep users engaged and motivate them to do more physical activities. Every time someone logs a workout, they are awarded points, which helps increase their level, there are also quests, achievements and challenges, these elements can incentivize users to exercise more often.

Fitocracy also has a big social component, there is a leaderboard where users are ordered by points, and a feed which shows the activities of other users allowing them to interact with each other.

2.5.2 Marketing: Starbucks Rewards

Starbucks has a mobile application[4] that incentivizes customers to purchase their products in exchange for stars, when the user reaches a determined number of stars they can earn rewards which are free products on their stores. This gamified experience has levels, and points in the form of stars, when a user receives a high quantity of stars, they reach the gold level which will provide them with better rewards and a personalized gold card.

This gamification is an improvement over normal loyalty cards because it has elements that make the experience more engaging such as the levels, regular feedback informing users of their progress for the next reward, and desirable goals. Starbucks Rewards may rely on extrinsic motivation, but it has had success reaching a high number of users.

2.5.3 Programing training: CodinGame

CodinGame[5] is a web platform that has programing puzzles for users, allowing them to practice their coding skills in a fun way. They use game elements such as levels, achievements and leaderboards, and they have puzzles that are related to games which usually have some graphic interface of a mini-game

[3] https://www.fitocracy.com
[4] https://members.starbucks.com/
[5] https://www.codingame.com

and it requires the user to change the code to obtain the desired behavior. All their puzzles include tests, allowing the user to try the exercise as many times as they want, receiving immediate feedback each time, informing them which tests passed or failed.

This website also has another mode which has a puzzle that is given to a group of players that are all competing against each other trying to solve it faster, at the end they can compare their solutions which helps them learn and allows some cooperation even though it is a competition

2.6 Gamification in Education

Students often feel unmotivated to perform their schoolwork which can cause them to have poor results. Applying gamification to education can be a good strategy to add some appeal to their courses and keep the students engaged and motivated to participate actively in the learning process.

The process of implementing gamification in an educational environment has the following main steps:

1. **Understanding the students' characteristics**: To design a gamification experience we should know the characteristics of the participants and define the profiles of the different types of students to determine what techniques would be suitable for them. For this step it is important to know the context of the educational program, as well as details about the students such as age and skill, in order to design an educational program with tasks that require an adequate skill level [4, 15].

2. **Definition of learning objectives**: Educational programs can have multiple types of objectives: behavior goals are focused on encouraging students to alter their behavior in some manner, learning goals include gaining specific knowledge or skills, and general goals are specific tasks or assignments that the students have to complete. The educator should have clearly defined objectives for the course, since these will determine what content and activities will be included in the learning process and will affect the selection of game elements [4, 15].

3. **Creating educational content and activities for gamification**: The activities should be designed in accordance with the previously defined objectives and allow for students to retry them, so they can improve their skill. They should also be feasible but with an increasing difficulty for subsequent tasks [4].

4. **Adding game elements**: For this step it is important to decide how students' progress will be tracked, what are the rules of the system and how feedback will be provided. For these functionalities we can use elements such as points, badges and levels, in order to incentivize users to complete objectives allowing them to focus on bettering themselves and recognize self-achievement.

If the educator wants to promote cooperation or competition then they can also introduce social game elements such as leaderboards [4, 15].

Gamification can be an effective tool to improve the learning process, however it must be applied carefully to prevent possible negative effects to the students' education. "The accuracy and efficiency of applying gamification to the education program will depend on the thoroughness of implementing these steps" [15].

2.6.1 Examples

This section will present a few examples of gamification applied to education in various school levels.

2.6.1.1 JFDI Academy

At the School of Computing, National University of Singapure, a gamified system called JFDI Academy was created for a programming course [3, 15]. Prior to implementing it, the professor identified the main issues: students tended to start working close to the deadlines, there were a low number of assignments and the feedback was slow. To tackle these problems and make a well structured education program, they increased the number of assignments, the online curriculum was able to provide timely feedback using auto-grading and they added the ability for students to raise questions online. The game elements used were experience points, leaderboard and a storyline featuring missions and side quests.

This experience showed positive results, JFDI Academy was able to induce a consistent behavior from students. Most of them found the system to be helpful and that it encouraged them to finish assignments which they delivered at a faster pace. However, there were some who felt overwhelmed by the number of deadlines.

2.6.1.2 ClassCraft

ClassCraft is a gamification platform for elementary and secondary education inspired by Role-Playing Games (RPGs), allowing participants to form teams, each member with their own character [16]. They use elements such as Experience Points (XP), achievements, challenges, avatars, among others. With ClassCraft students can choose between three different characters that have different attributes. Then they are encouraged to collaborate with each team members to achieve a common goal, but they can also have some competition with the other teams. Each character has attributes such as experience and health points, which can be increased or decreased by the teacher as a form of feedback. As the students complete tasks, they get awarded XP representing their progress towards completion of the course.

2.6.1.3 Khan Academy

Khan Academy[6] is an educational website that helps students studying for many different courses, they provide video lectures and exercises for users to practice what they learn. This website uses multiple gamification elements: points are awarded when users complete lessons or exercises, they can also earn badges for completing specific goals, there are progress indicators for users to be aware of how into the course they are, and the exercises provide quick feedback.

The exercises on Khan Academy give the student the opportunity to try them multiple times until they accomplish their goals, giving immediate feedback on those that can be corrected automatically. This is especially useful in the programming exercises that show the new output whenever a change is made to the code and they also display hints about what should be altered for the challenge to be completed.

This website has a high popularity and is a successful learning tool. However, some of the gamification elements such as the points probably will not provide intrinsic motivation, since they do not relate to an underlying activity resulting in a hollow gamification. This could be improved by relating the points with well-defined goals [17].

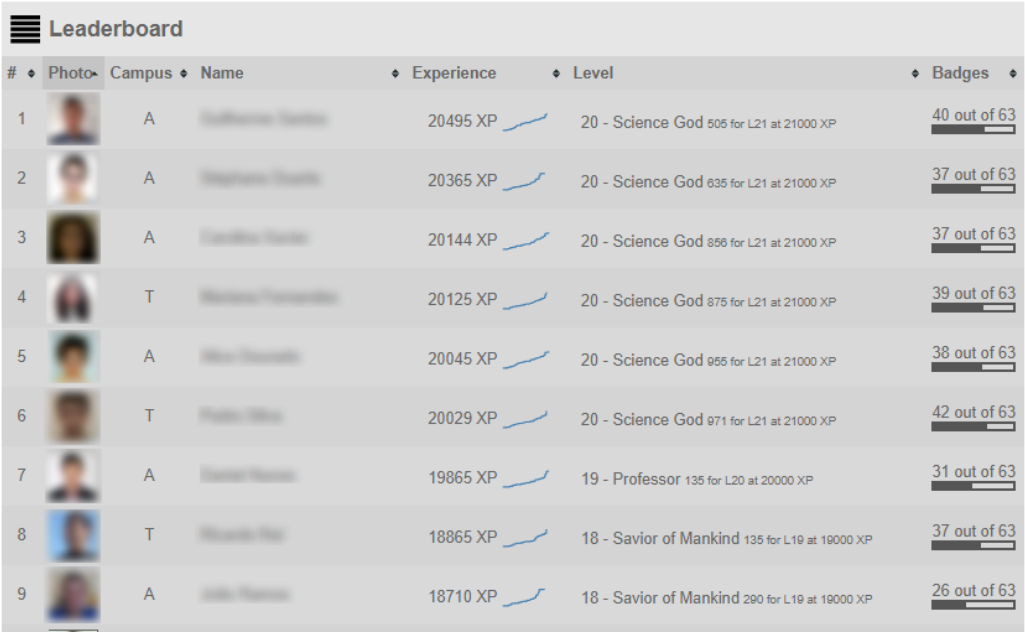
2.7 Learning and Gamification Systems

Something that is often used nowadays in education are VLEs, which are web platforms with resources that can improve the education process. Another similar concept is Learning Management System (LMS), internet-based systems that integrate pedagogical and course administration tools. They can be used as a complement to regular campus-based courses and they could even be used to develop fully online universities [18].

Moodle[7] , which stands for stands for modular object-oriented dynamic learning environment, is an example of an open source VLE/LMS that enables educators to create their own customizable website to use for educational purposes. This learning system has some gamification functionalities that can be used such as badges and levels. Moodle is used in MCP along with the Smartboards web application.

There are multiple others learning management systems including Blackboard, ATutor, Edmodo, and many more. Some of them have gamification features but that is not their main focus.

[6] https://www.khanacademy.org/
[7] https://moodle.org/



Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
1		A		20495 XP	20 - Science God 505 for L21 at 21000 XP	40 out of 63
2		A		20365 XP	20 - Science God 635 for L21 at 21000 XP	37 out of 63
3		A		20144 XP	20 - Science God 856 for L21 at 21000 XP	37 out of 63
4		T		20125 XP	20 - Science God 875 for L21 at 21000 XP	39 out of 63
5		A		20045 XP	20 - Science God 955 for L21 at 21000 XP	38 out of 63
6		T		20029 XP	20 - Science God 971 for L21 at 21000 XP	42 out of 63
7		A		19865 XP	19 - Professor 135 for L20 at 20000 XP	31 out of 63
8		T		18865 XP	18 - Savior of Mankind 135 for L19 at 19000 XP	37 out of 63
9		A		18710 XP	18 - Savior of Mankind 290 for L19 at 19000 XP	26 out of 63

Figure 2.2: Leaderboard of MCP's students

2.8 Multimedia Content Production

MCP is a gamified MSc course in Information Systems and Computer Engineering at Instituto Superior Tecnico (IST). This course has been the subject of studies analyzing the impact of gamification on students' behavior and identifying player types [5, 19, 20]. Part of the evaluation is done in classes, and there are also online assignments delivered through Moodle. The students are graded with XP that translate into grade points, they can check their scores in a web application that has a leaderboard (Fig. 2.2) and profile (Fig. 2.3) pages with information about each student, their badges and the skill tree (Fig. 2.4).

The leaderboard lists all of the students by XP and allows them to compare themselves to others. The badges require students to complete tasks such as attending classes, helping colleagues in the forums, and finding bugs in the class materials [21]. The skill tree is a set of assignments that can be done at any time, with the possibility of multiple paths being followed to achieve the maximum allocated XP. The design of game elements for the course aimed to provide the three needs of the Self Determination Theory. The feeling of relatedness could be boosted by the leaderboard and the interactions between student on the online forums, to increase feelings of competence students are provided with frequent feedback about their progress. Autonomy is provided by allowing choices to students, allowing them to pick their preferred path in the skill tree and choose which badges to complete [5].

Barata et al. [20] structured course activity into meaningful challenges and quests and conducted a study comparing the first three non-gamified years of this course with the following two gamified years.

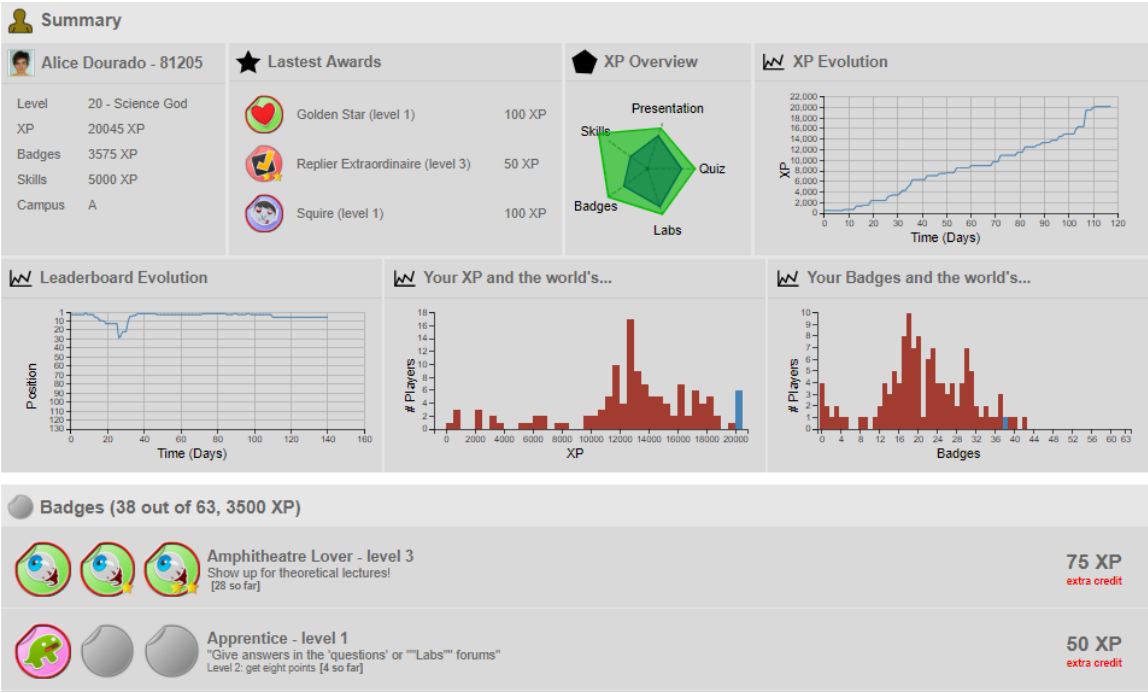


Figure 2.3: Student's Profile page with information and badges

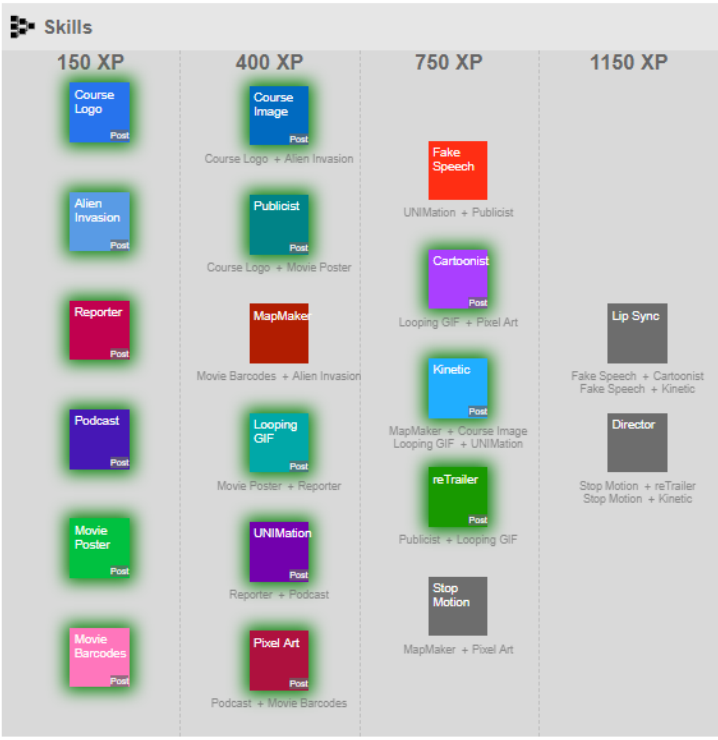


Figure 2.4: Skill Tree of a student

The authors purpose was to assess how gamification impacted the learning experience of students. Barata et al. observed that students were more proactive and participative as compared to our previous regular version of the course, as they considered to be more motivated and interested than in other non-gamified courses. In particular, results suggest that students were more participative and engaged in the gamified course, since they downloaded more course materials and had higher participation in the forums. Moreover, at the end of the semester they expressed that the course was more motivation and interesting than non-gamified courses via questionnaires. The first gamified year had an average grade slightly lower than the previous years, possibly due to the uneven distribution and low quantity of assignments. However, in the second year after some improvements to balance the distribution of tasks, the grades had the highest average with a reduced discrepancy. With this in mind, Barata et al. suggest that gamification can improve grades if students are given many opportunities to succeed.

However, it immediately became apparent that gamification impacted different students in different ways. Using machine learning techniques we have identified different student types [5, 22], based on XP accrual over time. Alas, while two student types remained constant over the years (the Achievers, students who went after every badge and performed the best, and the Underachievers, students who seemingly only did enough to pass the course), different clusters in between were observed, hinting at a deeper truth. Indeed, given the amount of data collected throughout the semester, it is possible to use more advanced Educational Data Mining techniques to better understand and predict student behaviour.

In the last decade, there have been several efforts to accomplish both those goals, mainly through the exploration of the data collected through intelligent tutoring systems and lately through online platforms. In both cases, the data was mainly explored through simple formulas (as in [23]) and tabular tools (see for example [24]), with particular interest on Bayesian networks and decision trees [25], despising its intrinsic temporal nature. Exceptions have been the exploration of the sequential nature of data to help on students profiling [26–28], and the anticipation of results prediction based on the exploration of temporal precedences among data [29, 30]. To our knowledge, there were only a few attempts to use sequence classifiers [31] and clusterers [32] with promising results, that should be thoroughly explored in the gamifying context. This thesis has the goal of providing the gamification software that will help the ongoing project GameCourse, which aims to bridge that gap of researching gamification effectiveness while adapting the experience to student's needs according to data collected with the techniques mentioned above.

3

GameCourse As Is

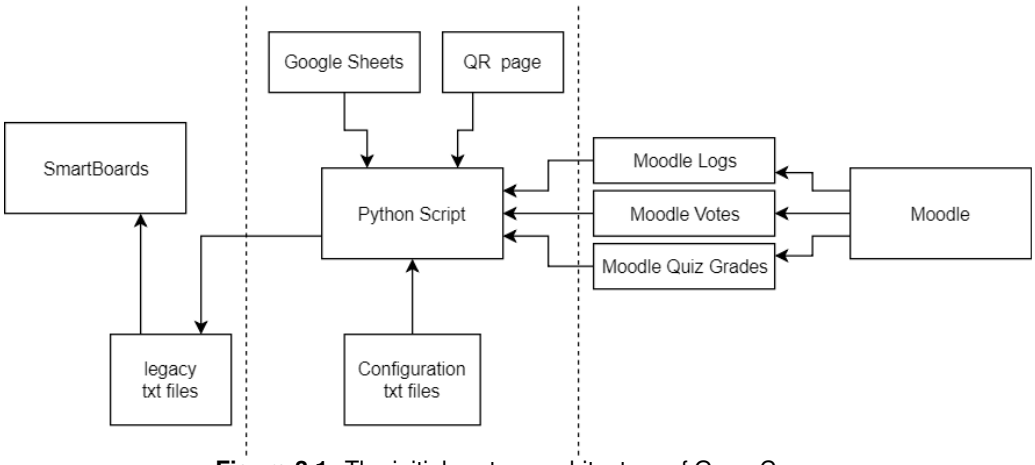


Figure 3.1: The initial system architecture of GameCourse

For the first years the game elements of the course were displayed in static web pages generated by a script that had to be manually run regularly [19]. In 2013 there was an attempt to create a web application to replace the old static pages as a thesis project of a MSc student, but this project was not concluded [33]. Then, in 2016, André Baltazar developed SmartBoards [34], a web application that showed the leaderboard and profile pages with an improved look and allowed for customization of these pages with the possibility of showing different Views to specific users. However, while the student front-end is working satisfactorily, the remaining parts of this gamified experience from collecting student data to awarding XP is still done piecemeal, by a multitude of disparate elements. Currently there is still a script that gathers and processes data from multiple sources. This information is then loaded into the SmartBoards system. The problem with this scrip is that it is a file of considerable length with a large quantity of hard-coded information, which means that for the system to be used on a different course with its own specific game elements, it would be necessary to edit a large quantity of code on this file, requiring knowledge of Python programming.

At the start of this project the gamification system of the MCP course (Fig. 3.1) used the SmartBoards platform programmed with PHP 5.6 and JavaScript, a Moodle website (version 2.6), multiple PHP scripts and a Python script to gather data from the Moodle logs, Google Sheets, some configuration files, and the QR list which contains the data for the ”Class Participation” badge. To update SmartBoards’ leaderboard, a faculty member must run the Python script, then replace some files with those generated by the script, and finally run a PHP script that updates the new information. If they want to update the student list, they need to add the student’s information to four different places (a configuration text file, Moodle’s list of students, the QR database and Google Sheets) and do the same steps to update the leaderboard but with two extra PHP scripts. This system has too many dispersed parts that lack cohesion and there is a large quantity of hard-coded information in its files.

The Python script collects the information necessary for the SmartBoards system. One of the sources of data is the QR page, which has the list of class participation by student for the "Talkative" badge[1]. The script also gathers data from Google Sheets, which is where the grades for all the evaluation components that are not automatic are listed, such as laboratory and presentation grades.

The Moodle page has some PHP files that gather specific data which is necessary for SmartBoards. Data includes activity logs, quiz grades, and rating of posts on the forums. Main applications for Moodle in this course are:

- provide resources (theoretical slides, laboratory guides, among others);
- delivering assignments for the skill tree;
- participating in the forums which can help winning badges; and
- doing quizzes.

The remaining information for the script is in the configuration text files, which list the students, faculty, achievements, skills for the tree, levels and awards. By running the aforementioned Python script, new text files are to be used by SmartBoards.

Thus, setting up Smartboards can be difficult due to the many steps for the installation. In short, to use SmartBoards you need to:

1. Update the *students.txt* and *teachers.txt* files;
2. Run the Python script;
3. Set up a Fénix application to allow students to log in using their IST credentials;
4. Update the *config* file with database and Fenix information;
5. Install dependencies (Jison and Less);
6. Run the *generate.sh* script, which processes the less and jison files;
7. If there is a file called *setup.done*, delete it;
8. Go to the SmartBoards page to create a course;
9. Run the *loadLegacy.php* script, which will process the information given by the Python script and insert it into the system;
10. Update cookie information for the *updateUsernames.php* and *downloadPhotos.php* files, which need access to Fénix;

[1] Whenever a student participates in class, they are given a QR code that will lead them to a form. After filling it in, their class participation is registered in the QR page.

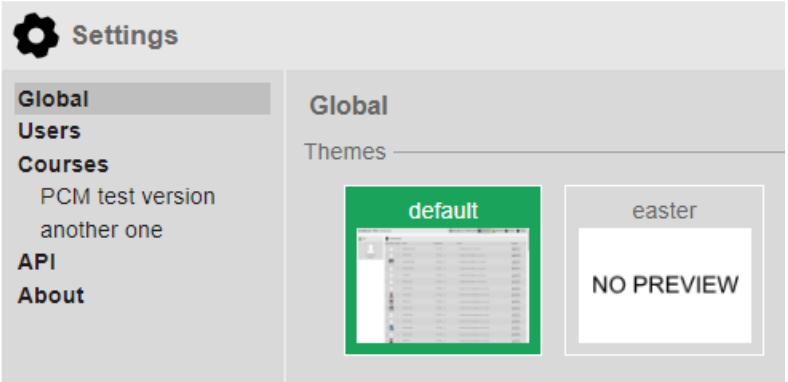


Figure 3.2: The main Settings page in SmartBoards.

11. Run these files; and

12. Configure the Views from the pages in Smartboards.

SmartBoards [34] is accessible to students and faculty by showing leaderboard and profile pages, as described in Section 2.8. Additionally, there is a back end for the faculty that has settings allowing them to configure the platform. The main settings page (Fig. 3.2) has options to change the theme, and manage the courses and users, with limited functionalities.

For each course there is another settings page (Fig. 3.3), this is where modules can be enabled, the Views can be configured, and the users' roles can be altered. While modules are responsible for the extensibility of the system, i.e. they define pages and game elements, Views are responsible for defining the elements of each page and adding differentiation. Modules can define Views for each page they provide and the latter can be configured (Fig. 3.4) in the settings, using the editing interface and an expression language to access the data of the system. Moreover, it is possible to define different Views based on the role of the current user.

This system [34] uses a key-value database that hashes its information and stores it inside a relational database as binary objects. This means that although it is using a relational database is not actually organizing its data in a relational manner, since it only has three tables without any relationship between them and the data stored is not human-readable. The data can be accessed from modules and from the system on the Views editor with a data schema to facilitate the use of Views. Currently the pages can take a considerable time to load because the database has latency issues and due to the fact that the system is running on Sigma, a cluster available for the academic community of IST. Taking into account all preliminary work, it is clear that there is no system or platform to allow an easy and fast installation and configuration of a gamified environment.

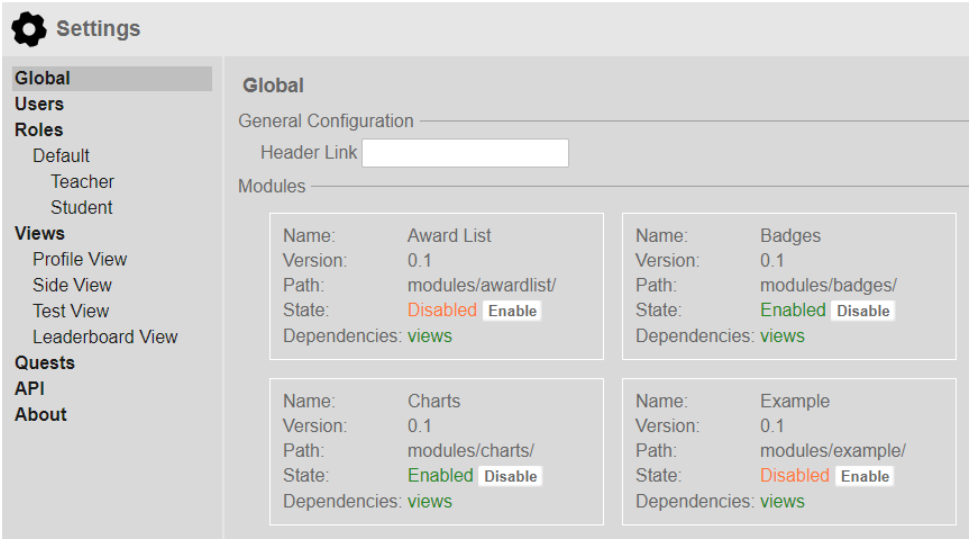


Figure 3.3: The course Settings page in SmartBoards.

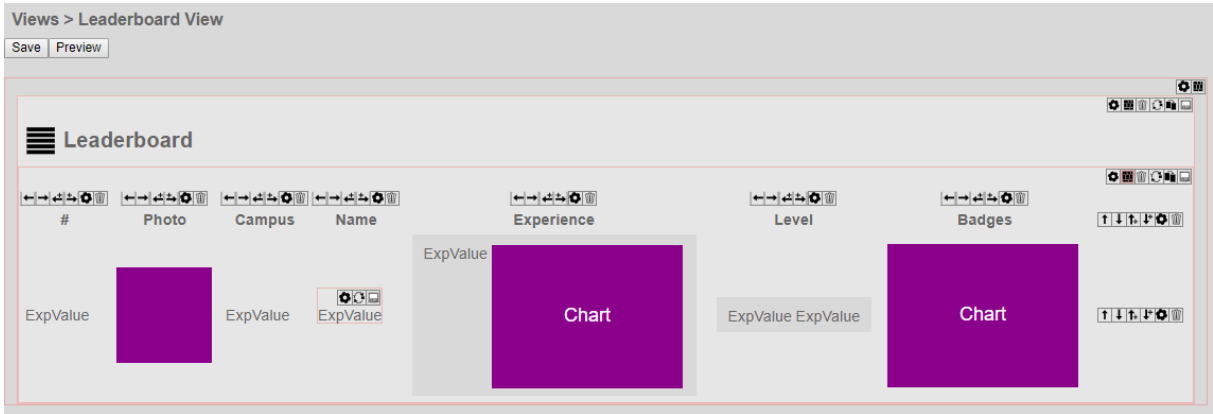


Figure 3.4: Configuration of the leaderboard View.

4

Development

Contents

The development of this project was initially focused on enhancing the SmartBoards system, by fixing its main issues and improving load times of its pages. Meanwhile, there was also some assistance given with tasks related with Moodle and its maintenance. After the new version of SmartBoards reached a stable version, we started working on planing GameCourse, with a new database and improvements to the expression language used in Views. For this project, several parts of the GameCourse system were improved, but the main focus was on the SmartBoards platform which will be the base of future versions of GameCourse.

This Chapter contains a detailed description of the work done to develop the current version of Smartboards and the remaining parts of the system. We will start by explaining initial changes made to the system, the following section will describe the architecture, then we will go into detail about users, modularity, Views, and the expression language that was created.

4.1 Preliminary Work

The first stage of the development consisted of fixing SmartBoards' latency problems and improving over its flaws, while also making improvements in the other remaining parts of the system. This new version was deployed in the middle of the second semester of the 2018/2019 school year, while the MCP course was taking place, and was regularly updated whenever new features were made or students reported bugs.

4.1.1 System-wise

As mentioned before, most of the work of improving the old gamification system was developing the new SmartBoards platform. However, there were also other tasks done to improve the remaining parts of the system and assist in the GameCourse project: recreating a questionnaire for the MCP students, extracting data from Moodle, and updating Moodle. The following sections go into detail focusing on each of these parts.

4.1.1.1 Learning Styles Questionnaire

For the GameCourse project, it was necessary to gather data from students in order to make a pedagogical analysis. Students answered questionnaires on Moodle and in class, including the Index of Learning Styles (ILS)[1]. This instrument has 44 questions that classify a person in four dimensions, each with two opposite categories (Active/Reflective, Sensing/Intuitive, Visual/Verbal, Sequential/Global). This questionnaire was available online but we wanted to have results with every answer given to each question,

[1] https://www.webtools.ncsu.edu/learningstyles/

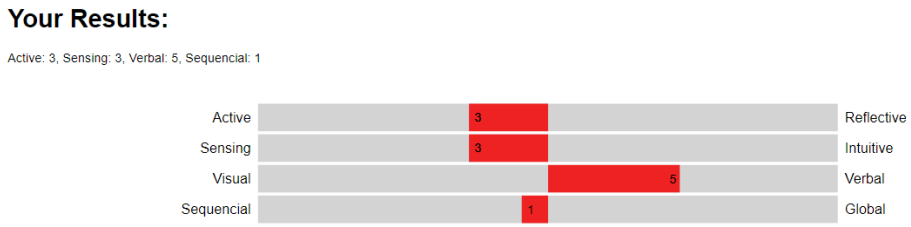


Figure 4.1: Results of the Questionaire in our ILS page.

instead of just the final categories assigned to the student. Therefore, the questionnaire was carefully examined to determined exactly which questions contributed to which dimensions. Then it was recreated with our version producing the same results as the original. We created a page with the full questionnaire which would then show the results with a graphic (Fig. 4.1) similar to the original. This page had a simplified look, but the results were all stored in a database and thus available to be analyzed. Afterwards, we extracted all data with a script that put the information on CVS files, listing every answer given to each question and the resulting dimensions assigned to each student.

4.1.1.2 Extraction of Moodle data

Another dataset that was required for the GameCourse project were logs with all messages sent on the Moodle platform. To gather this data, we created a script inside the Smartboards system. The reason for this location was so it could use functions to access the database that was already defined in this system and currently Smartboards and Moodle share the same database. This script was responsible for going through data of each course, their forums, their discussions, and their posts and extracting it. All data was put into folders with a similar organization to Moodle and each individual post had its own JavaScript Object Notation (JSON) file. Post messages were initially copied without any alteration which meant that they contained HTML tags. After a request to remove the tags and other unwanted parts of the text, the script was modified so it deleted most HTML tags, and replaced images and weblinks with keywords, using Regular Expressions.

4.1.1.3 Updating Moodle

The Moodle website used for the MCP course was in version 2.6, which is very outdated and could have safety vulnerabilities or other issues, since the most recent version is 3.7. Thus, it needed to be updated. We backed up all the data and updated Moodle on a local copy of the website to assure that there were no issues. This allowed us to figure out the updating process which had to be done in a few separate updates because there are versions that need obligatory updates before going for a higher version. Thus, we started by updating to version 2.7, then to 3.2, afterwards we attempted to update to

most recent version. With this update, we had to uninstall the peer grading plugin that was developed to be used in MCP due to it not functioning in the version of Moodle that we installed in the end.

MCP's Moodle website is currently using version 3.2 because Sigma's cluster, where it currently is hosted, does not have the PHP requirements needed to update to newer versions. This Moodle should eventually be migrated to the Virtual Machine (VM) of the course where it would have the necessary requirements to be updated to the most current version.

4.1.2 SmartBoards

SmartBoards was the main focus of the initial stage of development. The goal was to fix its main issues so it could replace the old version being used in MCP. In particular, we optimized the database, moved the QR page inside the system, added Google authentication, fixed multiple bugs, and created configuration pages.

4.1.2.1 Database Optimization

The old system had a key-value database which functioned in a way that resulted in a considerably long time to load pages, thus, it needed to be optimized. This required an analysis of the system to determine how the data was previously organized and how it should be structured in the new database. There was a page with the data schema which clearly showed the structure of the data that was accessible through Views. However, the remaining data of the system, concerning all the configuration information, was not documented. Thus, it was necessary to examine the code that dealt with Views in order to understand how this part of the data was organized. To represent all data of the new SmartBoards, we designed an entity-relationship schema. Afterwards we created a relational database using MySQL. A new class was created which is responsible for all interactions with the database, establishing the connection and sending queries using prepared statements. All code that stored or retrieved information was modified to use functions of this new class instead of the previous data source. One of the most significant changes was in the script which goes through all text files with data, such as awards, users and badges, and updates their information in the database. The other major change was the files associated with the expression language used to edit the Views.

The language allowed access to data while editing Views by writing expressions based on the schema of the key-value database, requiring the user to know the names of the arrays, maps, objects, and values existent in the database. For example, to get a welcome message with the name of the user viewing the page we would write **Hello** {**users[%viewer].name**}**!**, since *users* was a map of user objects with a *name* field. The new version of the language did not necessarily improve or decrease its quality. It only aimed to make the language compatible with the new database, since the language kept being a direct mapping of the database. Usability issues of the language were only addressed latter in

this project (Section 4.6). Part of the language remained the same as before and major changes were made in the way it could access data, it now required that table names and its parameters would be specified with the added possibility of using a natural join between tables by adding a plus (**+**) sign.

With the updated database there were already noticeable improvements in the load times, so this intermediate version of the system started being used in MCP, eventually the database and expression language were redesigned in what will be referred as the new version of SmartBoards.

4.1.2.2 QR page

In the previous system, the QR page was one of the data sources for the Python script which collects data for the system. The QR page was responsible for creating QR codes, storing class participations of each student, and displaying data of all the registered participations. These functionalities were put into the new SmartBoards as a module, which created a page (Fig. 4.2) for professors where they can create QR codes and check the list of student participations, and of failed attempts at registering participations. With this module in the website, we can remove a part of the Python script and this data can be directly added to the database when a student registers their participation.

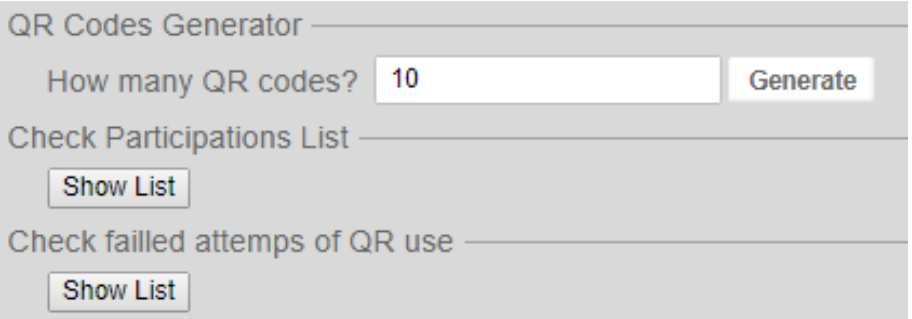


Figure 4.2: The QR page on SmartBoards.

4.1.2.3 Google Authentication

SmartBoards only authentication method was Fenix. Users logged in using their IST credentials, which is a method that works if all users are registered on that domain. However, if we want this system to be usable in other contexts, it is important to allow other methods of authentication. The need for other forms of user authentication also came up due to the need of access to the SmartBoards platform to members of the GameCourse project who were not from IST and needed a different way to access the website.

Therefore we added Google authentication using the Google API Client Library for PHP. Although the Fenix authentication is still the default method, pre-approved users can log in with Google by going to

a specific URL which will authenticate them. Those who log in through the Google authentication page are automatically given permission to view all the courses without giving them functionalities of students or professors. The ideal version of the system should give users the choice of how they want to log in and allow more forms of authentication besides the ones mentioned.

4.1.2.4 Bug Fixes and Extra Features

While developing the new SmartBoards, there were many smaller changes which included bug fixes and new features that improved the system.

One of the first changes was the addition of a GIF to show that a page is loading, which provides a better user experience than the alternative of having a blank page that does not inform the user whether something is actually loading. We also fixed a problem that occurred in the old version, which happened when a user left the website opened until the session expired, which led the page to stop responding. Now, the system detects if the session expires and reloads the page.

All Views had to be remade because the expression language used in them was altered. The recreation of these Views gave the opportunity to fix some bugs that appeared in them. The star plot on profile pages had an incorrect parameter and a bug that was impeding the process of adding and removing parameters from those types of charts. There was also a bug on the chart that displayed the evolution of the position on the leaderboard for each user, which was showing inaccurate information for users that were repeating the MCP course because it was using all their data in calculations instead of just data of the current course. Part of the information displayed in Views was not properly ordered and, in some places, wrong data was shown. We also added a functionality to extract templates with the press of a button (previously they had to be manually extracted), so they can be used as pre-defined templates provided by modules. All of the newly recreated Views were defined as templates and then extracted and added to their respective modules in order to automatically add them to the system whenever the module is enabled.

Next, the access to courses was restricted to users of those courses and the system administrators, which means that students can no longer see the leaderboard and other data of the previous years' courses. This is important because this system could be used by a multitude of different courses simultaneously and we do not want students of a course to be able the see grades of students of all of the other courses.

Furthermore, there were also some bugs on the View editing page that were fixed. The most problematic one was that whenever we went to the settings of a specific element of the View, it would lose its scope, which resulted in the impossibility of then deleting that element and any attempt at removing it would delete one of the other elements. Another issue happened in the settings of an element, which is in an overlay that can be closed by clicking on the close button or anywhere outside the overlay. We

noticed that it was also closing if there was a mouseup outside the overlay, which can be a common occurrence when users are selecting text and drag the mouse outside. Now it only takes into account mousedown events. There was also a problem with the square that exists in expression fields which is green if the syntax of the expression is valid and red otherwise. In some cases of more complex expressions that were valid, it was incorrectly displaying the red square, which could cause users to assume they did a mistake when they did not.

In terms of front-end, there were not any significant changes, we just added padding on top of each element of the View editing pages to allow space for their toolbars. This replaced the previous padding that only appeared when the mouse was over elements, which caused those elements to jump around, making it difficult to press buttons. There was also some additional padding added to elements in tables that had partially occluded toolbars.

4.1.2.5 Configuration Pages

One of the objectives of this project was to make the system configurable without the need to alter text files and run scripts. However, this would require the completion of the rule system that would replace the Python script, responsible for gathering data from multiple sources and updating text files. In spite of being supposed to finish its development while the project of this thesis was ongoing, it is not yet completed. Given that this system is not yet finalized, the configuration text files must still exist since they are used by the existent script, but, even with this restriction, it is possible to create configuration pages.

There are text files with information about students, faculty, levels, badges and the skill tree, which is data that should be editable in a more convenient way. Therefore, we created configuration pages (Fig. 4.3) for each of these files. These pages are responsible for updating information in the files and in the database, since both of these should always have the same data, because the text files will affect information produced by the Python script regarding awards that users received. For now, the editor is a simple textbox in which data can be written in the same format as the text files. This textbox comes pre-filled with contents of the respective text file and it replaces all data of that particular configuration page with whatever is written in the box. The page also displays information in the database, allowing, the user to see the result of their changes in the configurations.

With the addition of these functionalities, we could remove part of the contents of the script responsible for updating SmartBoards data, which would always update information (such as badges data) even though it would probably only need to be updated at the beginning of a course. Now, the script no longer updates data that has configuration pages, and thus it became slightly faster. We also added the usernames of users to their text files which means that the *updateUsernames* script (which required Fenix cookies to access a page of the course that listed all users with their usernames) is no longer

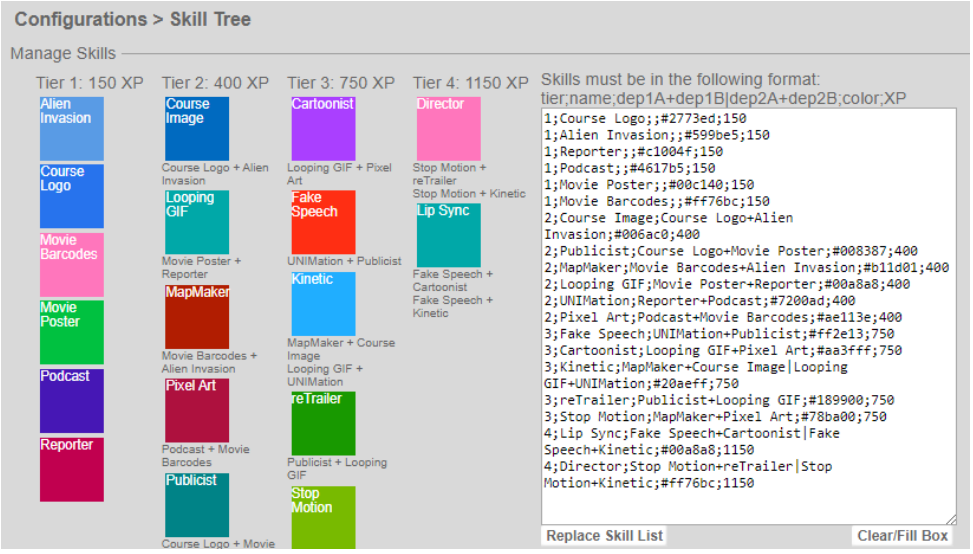


Figure 4.3: Configuration Page for the Skill Tree.

necessary.

In the previous version of the system, there could be multiple courses, but it did not allow a way to conveniently have multiple active courses that needed to be regularly updated. The problem was that when SmartBoards was updating, it always looked for text files in the same folder which was specified in the *config* file of the system, thus, to have two concurrent courses their administrators would always need to change the configuration before running the update script, and, if they forgot, the update would put incorrect data in their course. Currently, each course has a subfolder with its own configuration text files. The course folders will be useful even after retiring the Python script and no longer needing these files, because these folders can also store any other file of a course, such as user's photos, data for the skill tree, images of badges, and other relevant information that may come up.

4.2 Architecture

The original plan (Fig.4.4) for this project would have GameCourse be an entity independent of external scripts with all its information automatically updated. However, this would not be achievable because of the lack of a configurable rule system to replace the Python script which has large quantities of hard-coded information. Therefore, the objectives shifted to focus on restructuring SmartBoards so we can have the base needed for future developments which is a step in the direction to reach the originally planned architecture. The current version (Fig. 4.5) is an intermediate step between the initial stage (Fig. 3.1) and the intended GameCourse. This is a version that still relies on the Python script but one of the external components (QR page) was moved inside SmartBoards, the Moodle website was updated,

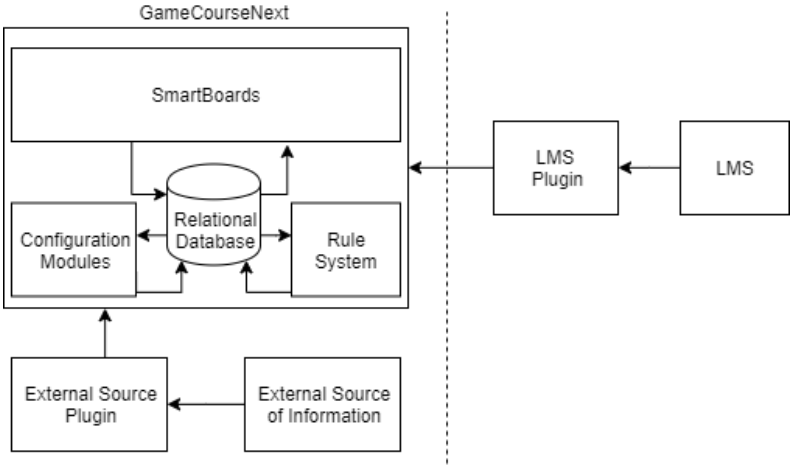


Figure 4.4: Intended Architecture for GameCourse.

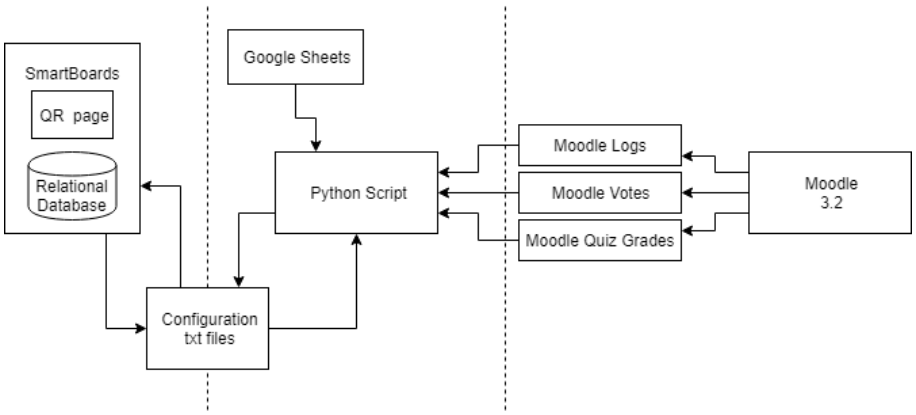


Figure 4.5: Architecture of the current GameCourse system.

the configuration files can be altered by SmartBoards, and there is a relational database.

In order to achieve the objectives, SmartBoards' database was redesigned so it could have all the necessary data in an organized manner. Figure 4.6 shows the diagram for the basic scheme solution of the system. A more in-depth diagram is presented in Appendix A. This system is versatile, allowing professors to create and customize pages with the possibility of defining Views for specific types of users. The language used to edit Views is organized in a similar manner to programming languages, instead of being depended on knowledge of the database, which should make it easier to learn. Faculty can also change configurations of the system directly on its interface. Another quality of this version is scalability since now it is possible to have multiple active courses at the same time. Professors can choose which modules they wish to enable and, if they want to add new functionalities, they may create more modules for that purpose.

As the second semester was ending, this version started being planned, and after we reached a

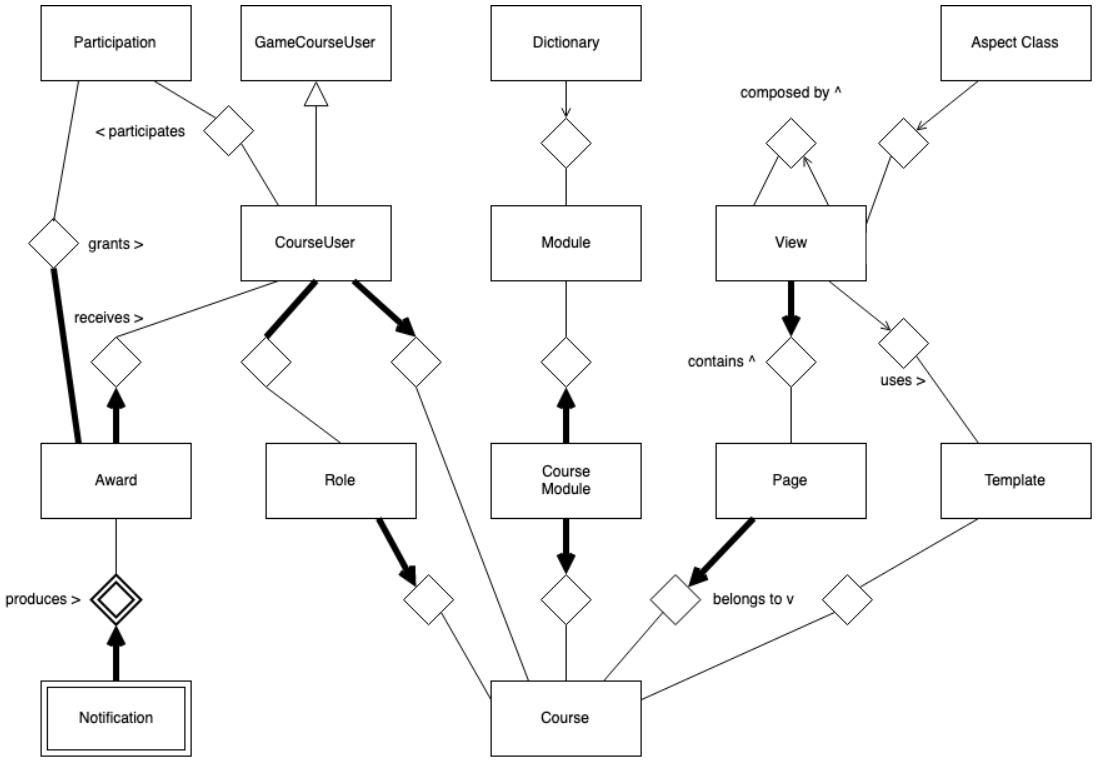


Figure 4.6: Entity-Relationship Model of GameCourse.

consensus on how everything should be organized the development started. The new GameCourse system has everything from the previous developments, but now with extra features and the restructuring of the database.

4.3 Users

In this system there are GameCourse Users who have basic information such as *username* which allows them to log in the system. If someone is just a GameCourse User they will not have access to any course, unless they are administrators. The other type of user is the Course User which is a GameCourse User who is assigned to a course. These users can have roles within that course, and may receive awards for work they perform in the course.

Currently, the existing default roles within a course are Student, Teacher, and Watcher. However, faculty members of courses can configure the roles in the settings, where they are able to create new roles, attribute them to users, change their hierarchy and add sub-roles. Therefore, it is possible to add any custom role to students, allowing us to use roles to indicate information such as learning styles. Afterwards, we can adapt pages to specific learning styles using Views. While the Settings page of the system is only available to GameCourse Users who are administrators, the Settings page of courses

is accessible to administrators as well as Course Users with the Teacher role. Roles have a hierarchy which is useful for cases where an user has multiple roles which could affect what they see on a page, so the system will choose their highest ranked role. For example, let us say we had a sub-role for students with a visual learning style, another for verbal learners, and a higher ranked sub-role for students with a current grade bellow what is typically expected which can danger of failing the course. If each of these roles had a different version of the profile page, whenever a student went to view it, they would see the version with the highest ranked, most specific matching role. So a student who is a visual learner and in danger of failing would see the profile for students who may fail the course, because that role has a higher rank. This could be used to show these students a warning telling them that they are falling behind and thus, should perform more work to ensure they get a passing grade.

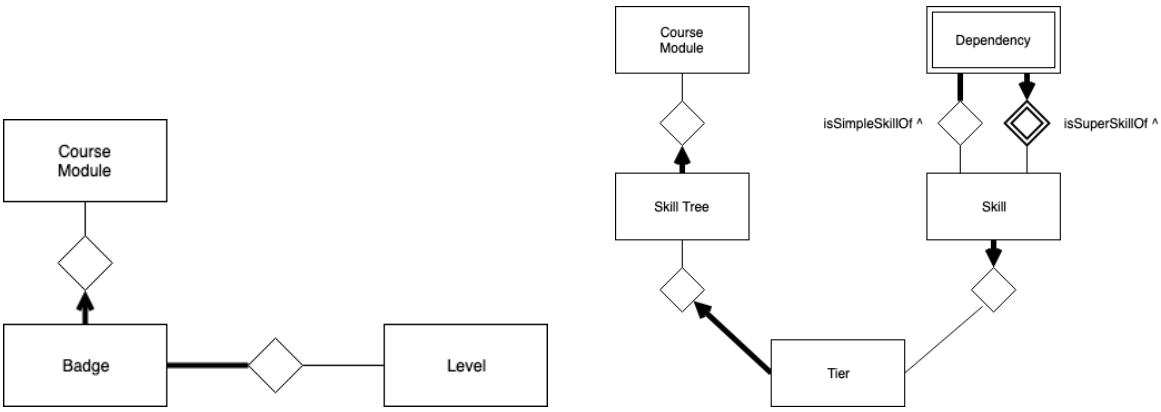
4.4 Modularity

An important part of SmartBoards are the modules which are a way for users to extend the functionalities of the platform by adding a few files without having to change the code on the remaining parts of the system.

Modules are created by adding a folder with the name of the new module within the *modules* folder. In this folder there must exist a "*module.moduleName.php*" file with the basic module configuration (more details in Appendix B), and it may include any other PHP, JS or CSS files that may be useful. Afterwards, modules will be recognized by the system and can be enabled in the Settings of a course by pressing the *Enable* button. It is possible for a module to depend on others which would mean that it could only be enabled after the one it depends on.

Modules can add many functionalities. They may be responsible for adding tables to the database, in which case they will be added when the module is enabled in a course and will be deleted if the module is disabled on all courses. An important use of modules is to add vocabulary to the expression language used in Views which will be explained in detail in Section 4.6. The language is divided into libraries with their own namespaces. A module may add functions of the language to an already existing library or create new ones. Modules can also be used to provide pre-defined View templates, create pages, and add them to the navigation bar. These functionalities can be manually configured in the system but it could be helpful to have them appear automatically by enabling a module.

There are pre-installed modules in the system which are: Views, badges, skills, XP, charts, QR page, profile, and leaderboard. For example, the badges and skills modules provide tables (Fig 4.7), vocabulary, templates, and CSS code. Unlike the previous versions, in this one, the XP of a student is not a parameter of a course user because XP is a module that can be disabled, thus, the value of the XP or level of students can only be calculated if this module is enabled.



(a) Badge Module Entity-Relationship Model. **(b)** Skill Tree Module Entity-Relationship Model.

Figure 4.7: Entity-Relationship Models of data provided by Modules.

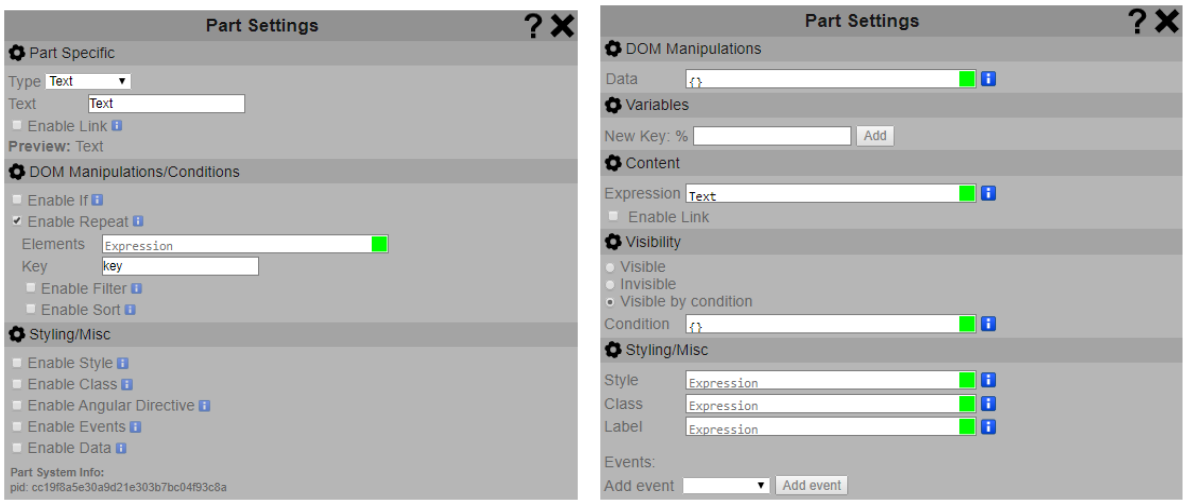
4.5 Views

Views are contents which are displayed to users in the SmartBoards platform. A View is composed of other Views. The most basic Views are images or text elements, these can be inside tables or blocks, and together all these individual Views can become more complex Views such as a leaderboard. A View can be associated with Data which will provide information from the database, and if it contains a set of data it will cause the View to repeat itself for each element of the set. It also has many other parameters that define its content and look.

Another important feature of Views is their ability to show different aspects depending on who is looking at them. This allows a View with a version for Students and another for Teachers, and it could even be used to differentiate Views between different sub-roles of students assigned to them according to information such as their learning styles. The aspects of Views may be differentiated according to the roles or IDs of users accessing the page and may also take into account the role of the user associated with the View which exists, for example, on the profile page. When a View is rendered, the system will first check the roles of the user to find the most compatible aspect of the View. If there aren't any matches it will show the default aspect

Views can be of the following types:

- **text:** An element containing text, which can be plain text, or an expression to be interpreted;
- **image:** An element containing an image. Its content is selected by writing a path to an image in the file system of SmartBoards. It may also use the expression language;
- **block:** An element that may have an header and it can have other Views as children;
- **table:** An element that represents a table, it can have children of the types **row** and **headerRow**



(a) Part Settings on the old system. **(b)** Part Settings on the new system.

Figure 4.8: Settings of an individual text part of the View.

which may then have their own children representing the individual elements on each row of the table;

- **templateRef:** A block containing the contents of a Template.

The Views can be edited in the Settings. This works in a similar way to the old system since there were no significant changes made to the front-end because that was not the purpose of this thesis. The main differences are in the expression used in the editing process, on the look of the settings of individual parts of the View (Fig. 4.8).

A View was multiple parameters that can be defined (Fig. 4.8(b)) in the editing process:

- **Data**: An expression that gets data to be used in the View or its children. If the **data** is a collection, it will be looped through and the View will be repeated for each of its elements. The use of this parameter creates a variable **%item** which can be used inside the View or its children to refer to the current element of the collection.

- **Variables**: A list of variables that are usable in the View and its children. It is defined by writing its name and then the expression that it represents. It is called with a % (percentage symbol) followed by its name.

- **Content:** An expression that represents the contents of a text or image View.

- **Visibility:** Radio buttons that define if the View should be visible, invisible, or only be visible if a condition is true.

- **Style:** Field for CSS code which will affect the style of View.
- **Class:** Name of an HTML class, could be useful if there is already some CSS behaviour for that specific class.
- **Label:** A label to identify the View.
- **Events:** Events that can trigger actions.

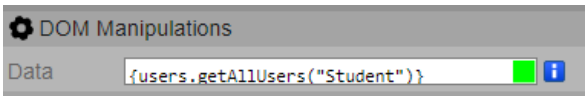
The current version of the View configurations still has many of the same fields as it did in the old system. However, some of them change their names or placement, and the fields are now all visible without the need to tick a box to enable them.

The **Data** and the **Variables** were moved to the top of the settings since these options create variables which may be used in the remaining fields. We also removed the sub-types that used to exist in the text and image Views. Another change was the removal of the Angular Directive field which was needed when we wanted to use events or JavaScript behavior, which means that users would need to know details about the system's code. Additionally, the functionality of the If and Filter fields from the previous version, which did the same thing, is now in the Visibility Condition. Finally, there was a Sort field which is something is now performed with the expression language. Some examples of Views will be provided in Section 4.7.

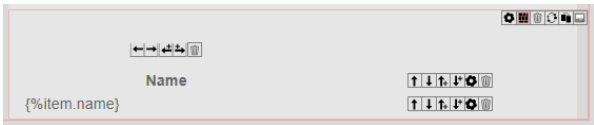
4.6 Expression Language

SmartBoards has its own Expression Language, which is used to personalize Views and interact with the database. While the previous version had a language that directly mapped its database, this new version was designed to be uniform, versatile, extensible, and to function in a similar manner to programming languages. We still kept some syntax rules such as the use of curly brackets, but everything else including all the functions of the language had to be created from scratch in order to follow our design. While editing Views, users can click on a help button that will open a page with an explanation of the View editing process as well as the list of available functions in our language.

This language is structured with multiple libraries, each with their own namespace, which provides their functions. Some of them are always available because concern basic concepts of the system such as users, courses and awards; others may be provided by Modules. To call functions, first we write the name of the library or an object provided by that library, then a dot (**.**), followed by the function name. There may also be arguments enclosed in parenthesis (**()**) and separated by comma (**,**). Any expression written inside curly brackets ({}) is an expression of the language to be parsed and evaluated. Everything else is considered plain text. This language has variables which are indicated with a percent symbol (**%**) followed by its name, and to escape this symbol we use **%%**.



(a) Expression to get all the Students in configurations of a Row



(b) Table which will display the name of users

Figure 4.9: Configurations to make a table that lists the names of all the Students.

An example of a basic expression would be "**Hello** {**users.getUser(%viewer).name**}**!**" which if the user viewing the page was called John, would be evaluated as "**Hello John!**". This expression started with plain text. Inside the curly brackets we called the function **getUser** from the **users** library, which receives an user id as an argument and returns a user object. The argument given was **%viewer** which is a variable that is always available, representing the id of the user who is logged in. Finally, we have **.name** which is a function to be called on an user object which returns its name. There are functions that return collections of values such as **users.getAllUsers** which would typically be used in the Data field. Most of these functions can be called without arguments but they usually have optional parameters that can specify conditions that the elements of the collection have to meet. For example, to create a table listing Students' names we could have a table View with a row containing Data of all the users with the Student role (Fig. 4.9(a)), and in an element of that table we would then have access to each user by using the %item variable (Fig. 4.9(b)).

Similar to many other languages, this one provides the simple mathematics operators for addition **+**, subtraction **-**, multiplication *****, division / and mod **%**. It also provides logic and bitwise operators, and **&&** (bitwise **&**), or || (bitwise |), negation **!** (bitwise ~) and bitwise xor ^.

As mentioned, each library provides their functions. However, there are some predefined functions that work on objects of any library, such as the following:

- **count:** To be called over a collection. Returns the number of elements it has.
- **crop(start,end):** To be called over a collection. Returns the collection only with with index between **start** and **end**.
- **id:** To be called over an object. Returns its id.
- **filter(key,value,operation):** To be called over a collection. Returns the collection only with with objects that have the parameter **key** that satisfies the **operation** with a specific **value**.
- **parent:** To be called over the variable %item. Returns the %item provided by the previous context.
- **sort(order,keys):** To be called over a collection. Returns the collection with objects sorted in the **order** ("ascending" or "descending") by the parameters in **keys** which has a string with the keys

separated by semi-colon (**;**). A **key** can be a parameter of the objects in the collection or it can be a function of the language as will be shown in examples.

The user has the freedom to obtain data and manipulate it how they wish, using the available functions, with the possibility of combining multiple functions. Here are some examples of how they could be used:

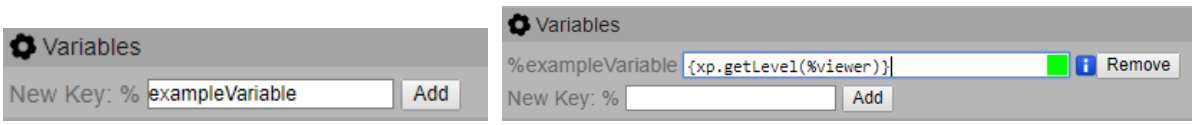
- {**awards.getAllAwards(%viewer).sort(”descending”,”date”).crop(0,4)**} **-** returns a collection with the viewer's five most recent awards.
- {**users.getAllUsers(”Student”).sort(”descending”,”xp.getXP; badges.getBadgesCount(%item)”)**} **-** returns a collection with all the users with the role Student, ordered by XP and number of badges.
- {**badges.getBadgesCount(%viewer)**} **out of** {**badges.getAllBadges.count**}**-** returns a expresssion saying the number of badges the logged user has out of the total number of badges. If the viewer had 10 badges and there were a total of 30, this expression would write ”10 out of 30”.
- {**users.getAllUsers.filter(”xp.getLevel(%item).number”,20,”==”)**} **-** returns a collection of all the users with with level 20.

4.6.1 Variables

Variables are represented with a percentage symbol (**%**) followed by their name. SmartBoards has already defined some built in variables:

- **%course** - Represents the id of the course that the user is manipulating;
- **%viewer** - Represents the id of the user that is currently logged in watching the page;
- **%user** - Represents the id of the user associated to the page which is being displayed, on pages such as the Profile Page;
- **%item** - If there is a Data, this represents the object that is currently being iterated in that View;
- **%index** - If there is a Data, this represents the current index while iterating a collection.

In order to create a new variable, while in the configurations of an individual View part, users need to write its name in the New Key field of the Variable section (Fig. 5.1(b)), then they press the Add button and write a valid expression (Fig. 5.1(a)). It is possible to define multiple variables in the same View and the variables can contain objects, collections or specific values.



(a) Defining a variable's name.

(b) Defining a variable's value.

Figure 4.10

4.6.2 Libraries

SmartBoards has several base libraries to provide read access to data in the system. Each library has its own namespace for its functions. Besides the built-in libraries, there are also others provided by modules and it is possible to add more vocabulary or libraries to the language by editing or creating modules. The difference between built in libraries or module libraries is just that the latter can be enable and disabled.

The following libraries are always available to the user without requiring any install:

- **actions** - Has all the information regarding actions to respond to event triggers. These functions should only be used in events.
- **awards** - Has all the information regarding Courses, CourseUsers, and Awards;
- **courses** - Has all the information regarding the Courses;
- **participations** - Has all the information regarding Courses, CourseUsers, and Participations;
- **system** - Has information regarding system specifications and some generic functions.
- **users** - Has all the information regarding GameCourseUserse, CourseUsers, and Roles.

There are also three libraries provided by the pre-defined modules which can be enabled:

- **badges** - Has all the information regarding Badges, and Levels. Also uses Award and Participation to get information about badges awarded to users.
- **skills** - Has all the information regarding Skill Trees, Skill Tiers, Skills, and Dependencies. Also uses Award and Participation to get information about skills awarded to users.
- **xp** - Has all the information regarding the Levels and uses Awards to calculate users' XP and level.

4.6.3 Events

Events are system functions that trigger whenever the user interacts with a View. SmartBoards supports several event types and each View can respond differently to each event following an expression written by the user. The following events are supported:

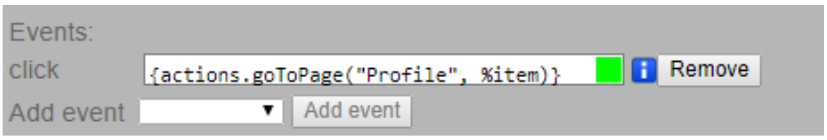


Figure 4.11: The definition of an Event on a View Element.

- **onclick** - The user clicks a View element;
- **ondrag** - The user drags a View element;
- **ondoubleclick** - The user double clicks a View element;
- **onmouseover** - The user moves the mouse over a View element;
- **onmouseout** - The user moves the mouse away from a View element;
- **onmouseup** - The user releases the mouse button;
- **onwheel** - The user rolls up or down over a View element.

Events can be defined in the settings of a View element by selecting an event from the drop-down menu and writing the function of the action that it should trigger, as seen on the example in Figure 4.11. When a user clicks on that element they will be redirected to the Profile page of the user in the %item variable. Events already existed in the old version, but only a few very specific events actions were available. The leaderboard module provided *goToProfile()* and badges provided *goToProfile()* and *showBadgeTooltip()*. These functions received as argument their own HTML element, so for example, to go to the profile when after clicking on a row of the leaderboard they would use *goToProfile(this.row)*. It was necessary to set the Angular Directive field to the one in the JavaScript files where these functions were defined in order to use them.

Currently the following functions, provided by the library *actions*, are available to specify actions triggered by events:

- **actions.hideView(label)** - Changes the visibility of a View referred by label to make it invisible.
- **actions.showView(label)** - Changes the visibility of a View referred by label to make it visible.
- **actions.toggleView(label)** - Toggles the visibility of a View referred by label.
- **actions.goToPage(pageName,[user])** - Changes the current page to the page referred by name. It may also receive an user object or id for pages such as the Profile Page.
- **actions.showPopUp(templateName,[user])** - Creates a template View referred by name in a form of a pop-up. It may also receive an user object or id.

- **actions.showTooltip(templateName,[user])** - Creates a template View referred by name in a form of a tooltip. It may also receive an user object or id.

These functions allow for the recreation of the actions available in the old system, but now there are many more possibilities of what can be done with them and there is no need to have a field called Angular Directive in the View's configurations. We can use the *goToPage* function to go to the Profile pages or any other one. The badge tooltip whose contents were previously defined in the code of the *showBadgeTooltip* function, can now be created as a template and used as a tooltip or pop-up as well as any other template. Finally there was a functionality that relied on JavaScript code on the badges module and specification of Angular Directives, which was to hide and show certain blocks in the badge list whenever the user clicked on a badge, this is now done using the *toggleView* function.

4.7 Examples

In order to show the potential of the new hierarchical View structure and the language can be used to generate complex Views, the following subsections cover three conceptual examples.

In these examples, we focus on three of the View's configuration elements: Data (top block, in the following diagrams), Variables (middle) and Content (bottom). As aforementioned, the Data object will be iterated and the current object will be in `%item`. Moreover, Variables are updated after each iteration, since they can depend on the `%item` being iterated. After it, the content is then rendered with the compiler evaluating each expression between $\{\}$.

4.7.1 Latest Awards

In Profile pages, there is a block with the latest awards received by the student (Fig 4.12). To make

Figure 4.12: Latest Awards View.

this View, we need a block View which will have the header, inside that block we need another View which will repeat itself for each award and show their image, description, and reward. We use a table

to ensure everything is aligned. The configurations of the table row will have the Data which will have an expression from the awards library that gets the last three awards. Then in each of the elements of that row, we will have access to the `%item` variable that will represent an award, and we can use it in the expressions to get the information we want. In Figure 4.13 we have the diagram of that row, and the contents of each element. This view also existed in the old version but it required the use of a function that received all skills in order to be able to render the pictures of the awards.

```
{awards.getAwards(%user).sort("descending","date").crop(0,2)}

{%item.renderPicture("type")} | {%item.description} | {%item.reward} XP
```

Figure 4.13: Latest Awards View Diagram.

4.7.2 Leaderboard Row

We need a block with a table inside it to represent a simplified version of the Leaderboard that appears in Figure 2.2. Similarly to the previous example, we will need to use the Data of the table row. Here (Fig. 4.14) we want to get all users who have the Student role, and they should be ordered by XP. The elements inside the row will have access to `%item` which will contain a user, and this variable will be used to get the ranking, picture, campus, name, XP, and level number and description. Since the level is needed twice we defined a variable %lvl with the level object.

```
{users.getAllUsers("Student").sort("descending","xp.getXP(%item)")}
%lvl = {xp.getLevel(%item)}

{%index+1} | {%item.picture} | {%item.campus} | {%item.name} | {xp.getXP(%item)} | {%lvl.number}-{%lvl.description}
```

Figure 4.14: Leaderboard View Diagram.

In the old system, the picture and level description are obtainable with more complex expressions, which will be discussed in Section 5.2. The leaderboard that appears on SmartBoards has some additional details including a few more columns, the use of Style and Class fields in the Views to improve their look, and an onclick event on the row View which will go to the profile of the user represented by `%item`.

4.7.3 Badges List

Lastly, to create a View with a list of Badges and their levels (Fig. 4.15), we need a block View with multiple Views inside it. This View (Fig. 4.16) needs a block to repeat itself for each badge, so we need Data to have an expression of the badges library to get all badges. There are multiple Variables defined

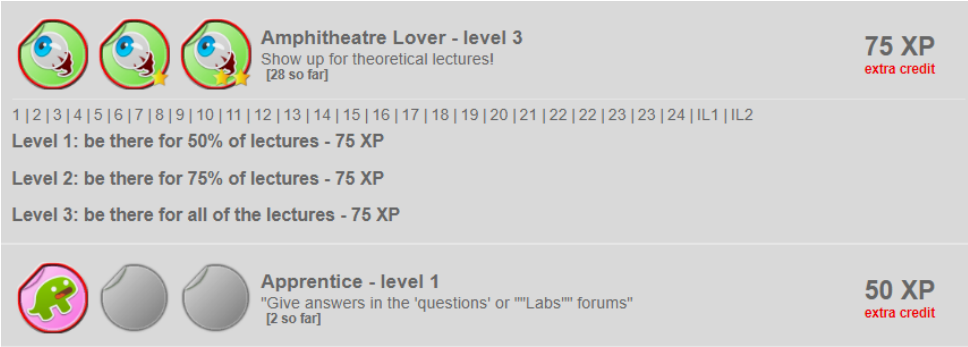


Figure 4.15: Badges List.

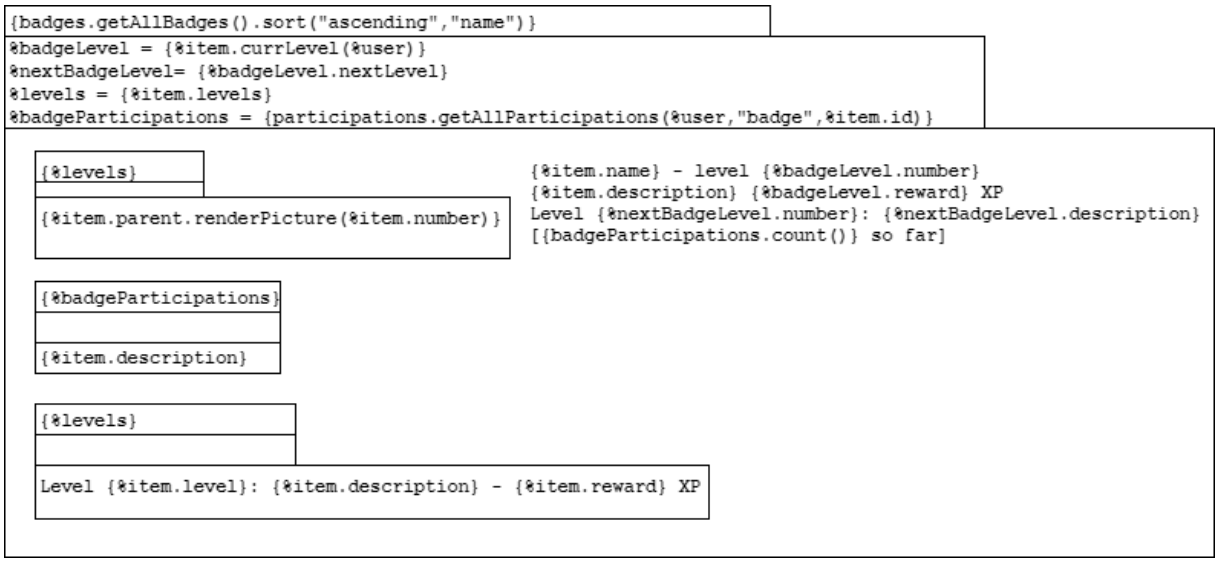


Figure 4.16: Badges View Diagram.

to avoid repetition in the expressions used. Inside this block we have multiple Views, some of them are simple text elements representing information such as the name, level, and description of the badge, and there are also other Views with more details such as the images for each level of the badge, the list of participations for that badge and the levels at the bottom of the diagram which is a View with %levels in its Data and the Content is an expression describing the level.

In order to have this View look like it does on the profile, it would also need some more Views and the use of Style and Class to improve the overall look. Additionally, there is a block view containing the participations and levels that appear in the diagram, which can be hidden or showed whenever the badge block is clicked on. This is done using the events and labels. These events used did not exist in the old version, so this functionality was previously achieved with JavaScript code in the badges module.

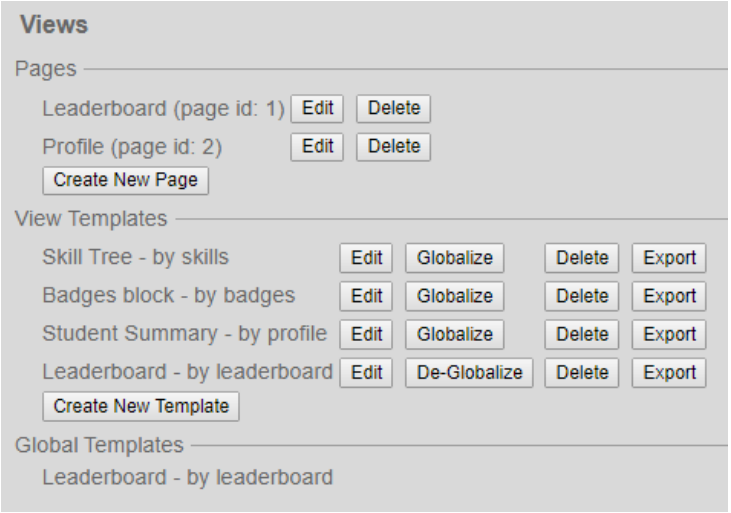


Figure 4.17: The View Settings Page, with Pages at the top and Templates at the bottom.

4.8 Pages

This system has Pages which contain Views. In the old version, pages could only be added and removed by enabling and disabling modules which provided pages, such as *leaderboard* and *profile*, and to add other pages we would have to create or edit a module. Now, while it may still be possible to automatically add them with modules, they can also be created in the Views Settings page (Fig. 4.17) by pressing a button and writing the name of the new page. When a page is created, it is automatically added to the navigation bar. It can still be useful to create a page with a module if we want to customize the URL of the page or its section of the navigation bar.

4.9 Templates

Templates are Views with names that can be used inside other Views. In the old system templates could only be created while editing Views by pressing the *Save Template* button on the toolbar, or could be automatically added by modules, and then they could be used in Views. There was no way to edit, delete or export them, and they could only be used by copy. Now, they can also be created from scratch in the Views Settings page (Fig.4.17). It is also possible to export Templates and add them to modules to be automatically added whenever the module is enabled. Templates are created inside the context of a course. However, they may be Globalized which allows them to be used in other courses. These courses will be able to use a copy of the Template, meaning that any alteration made to it will not affect the original.

Templates can be used in two different ways (Fig. 4.18): copy or reference. By copy works in the same way as adding and type of View element to the View being edited. When a Template is added

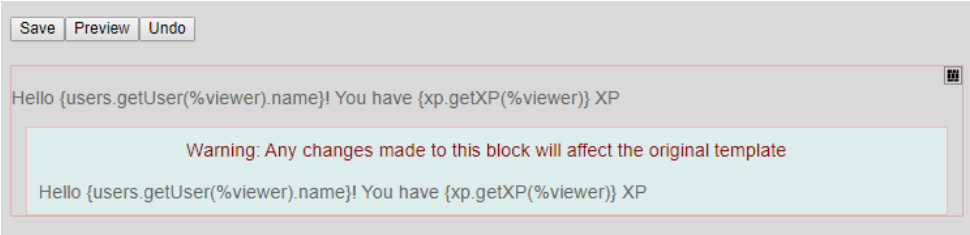


Figure 4.18: A text Template used by copy (above) and used by reference (bellow).

to the Page, the contents of the View of that Template with the closest aspect to the one being edited will be copied to the current View as a new element that appears inside it. Then it can be edited just like any other content of that View. They can also be used by reference, which will likewise result in the contents of a View of that Template appearing on the View being edited. However, the contents will be inside a block with a slightly different look and any saved changes made to its contents will affect the original Template. Consequently, any alteration made to the original template will appear in the template references. In Figure 4.18 we can observe a text template being used with the two methods.

4.10 Discussion

This chapter presented the current state of the SmartBoards system which is an in-between stage to GameCourse. It already includes one of the previous external elements (QR page), it has a relational database, and, while it still uses the configuration text files, their information can be edited directly in the platform. One of the major parts of the work developed was improving over flaws that existed in the previous version of the system. The other was developing important functionalities that give more versatility to what faculty members are able to do, using the platform without the need to alter code of the system. To reach the desired final stage of this system, it still needs improvements to its visual interface and the addition of a rule system.

SmartBoards continues to have Modularity, by having modules that can be enabled and will bring extra functionalities and vocabulary, but its not dependent on modules to perform simple task such as creating a page, adding a page to the navigation bar or use events. Pages can be created on the Settings and will automatically be added to the navigation bar, and events are all defined without depending on modules or needing to specify other parameters while editing Views.

Let us take for example the creation of a page ”Extra Info” and be able to go to that page by clicking on the user’s photo on the profile. In the current version we could add a new page in the settings page, and then edit the profile page so that clicking on the photo triggered actions.goToPage(”Extra Info”). In the old SmartBoards, we would need to create a module that would add that page, then we would add JavaScript code to the profile module, adding an Angular Directive with a function to go to the page, and

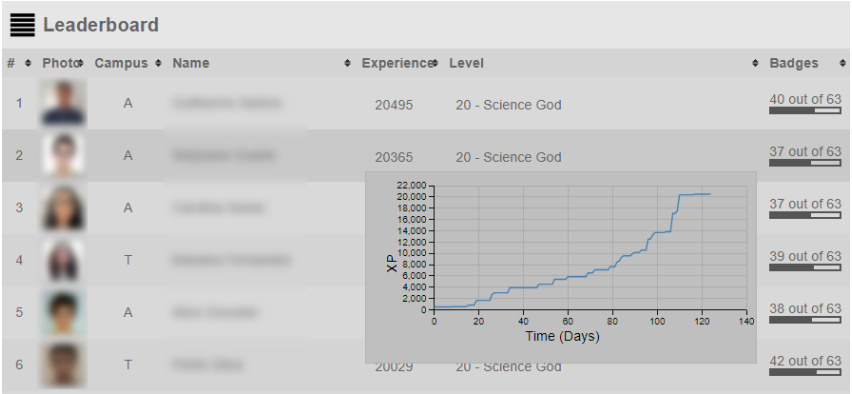


Figure 4.19: Leaderboard with a ToolTip showing a chart of XP Evolution.

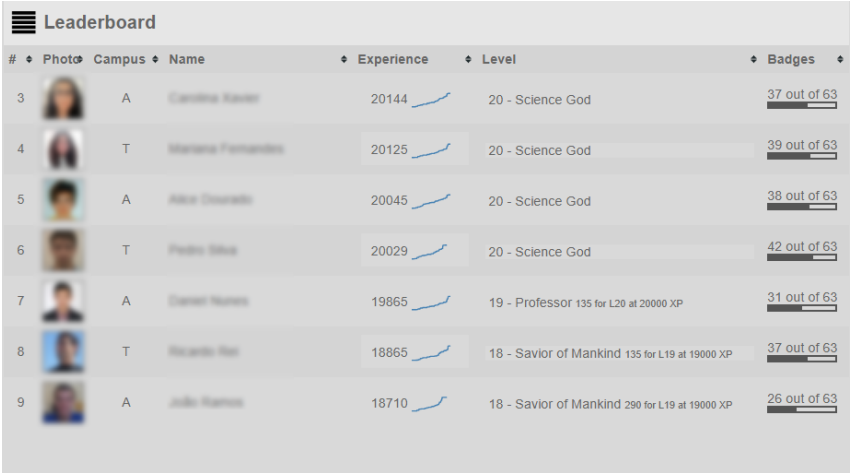
then we would edit the profile page in the platform, and in the photo add the Angular Directive and the event we just created.

Another example would be if we wanted to have a ToolTip on the leaderboard which showed the chart users' XP evolution, instead of current look where there is a small chart on each line of the board. For this we could simply make a template with that chart and then in the element which shows the XP we would add an event that did {**actions.showToolTip("templateName")**} and we would get the results shown in Figure 4.19.

We could also use the functionalities provided to create a different View of the leaderboard for users of specific roles, based on information such as learning styles of some other type of profile, which would have just the nearest neighbours in to the logged user instead of showing the full list. We would add an aspect to the leaderboard for users of the desired role, then would use get the list of users, but crop it so it only shows some of the students. This would create a shorter leaderboard (Fig. 4.20) that could be better for students who would get demotivated if they had to scroll down to find themselves in the leaderboard.

Furthermore, the updated language is also a big advantage of this version, since it is more intuitive and contains many functions that help users to easily access data. For example, if we have leaderboard that is using Data with users of the course to list them all and we wanted to add their pictures, right now we can simply write {**%item.picture**}, before we would write **photos**/{**users[%itemkey].username**}**.png**, which requires knowledge of the file system and the use of a parameter (username) which can not be obtained directly from the user in %item, making the expression more complex. A significant quantity of expression became more simple, and the more complex ones (such as the sort function) allow for more flexibility than what was previously possible.

To sum up, the developments made in this project allowed the system to have more versatility and flexibility by adding important functionalities that were previously impossible or required coding. The new database made Smartboards more stable. The extensible of system is also a very important quality,



Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
3		A		20144	20 - Science God	37 out of 63
4		T		20125	20 - Science God	39 out of 63
5		A		20045	20 - Science God	38 out of 63
6		T		20029	20 - Science God	42 out of 63
7		A		19865	19 - Professor 135 for L20 at 20000 XP	31 out of 63
8		T		18865	18 - Savior of Mankind 135 for L19 at 19000 XP	37 out of 63
9		A		18710	18 - Savior of Mankind 290 for L19 at 19000 XP	26 out of 63

Figure 4.20: Leaderboard only showing students near the logged user, as seen by the user ranked in fith place.

since it allows for the easy addition of new functionalities, and the modules along with the new configuration pages allow for a higher level of customization of courses. All these functionalities in collaboration with the new expression language made this system more user friendly, versatile and one step closer to becoming the desired GameCourse system

5

Evaluation

Contents

This chapter shows the evaluations performed on SmartBoards, which included Benchmark Tests to assess the load time of pages in the system, and user tests comparing the developed system with the original.

5.1 Benchmark Tests

Benchmark tests were used to determine if the update from the old version of SmartBoards to the one deployed in the middle of the semester, which had a new database using MySQL and had the objective of increasing page load speed, did indeed improve upon the problem it had with considerably high load times. The most recent version of the system, which has yet another design for the database, was also measured. For these test we created a leaderboard page with the exact same contents on all versions running on localhost and measured the time it took to load with a hard reload of the page and by going from another page to the leaderboard. For each case we made twenty measurements, calculated mean, median and standard deviation , then determined the percentage difference in relation to the old system. We obtained the following results:

Table 5.1: Leaderboard Load Time on different versions of SmartBoards

	Old Version		**Intermediate Version**				**New Version**			
	Hard Reload	**Normal Reload**	**Hard Reload**	**Difference**	**Normal Reload**	**Diff**	**Hard Reload**	**Diff**	**Normal Reload**	**Diff**
Mean	1.6287s	1.3448s	0.8835s	-45%	0.5007s	-63%	2.3308s	+43%	1.7943s	+33%
Median	1.6385s	1.353s	0.8745s	-47%	0.4975s	-63%	2.2855s	+39%	1.713s	+26%
Std. Deviation	0.0902	0.0503	0.0234		0.0202		0.3344		0.2004	

Afterwards, each of these samples was tested for normality using the Shapiro–Wilk test. The samples from the old version were the only ones with a normal distribution, therefore all comparisons between versions were made using the Wilcoxon signed-rank test. In order to prove that the intermediate version is faster, we compared its load times with the old version. The test comparing hard reload times test concluded that there were significant differences between the load time of each version ($p=0.000089<0.05$). For the load times without refreshing the page, the test also reveled significant differences ($p=0.000088<0.05$). Therefore we can conclude that the intermediate version of the system had significant faster load times than the old SmartBoards.

These results showed that this update improved the load time of pages of the system. However, this is still the intermediate version, and with the most recent developments the speed has decreased again, reaching even higher load times, due to higher complexity of Views and the need to calculate values such as XP. Therefore, the current system still needs to be optimized by creating caches of calculations made in the View rendering process.

5.2 User Evaluation

After finishing the development we performed user tests comparing the initial system with the one developed in this thesis.

5.2.1 Participants

User tests aimed to compare how well users performed in the execution of tasks regarding the View editing process. The functionalities tested were those which could be done in both versions of the system, focusing mostly on use of the expression language. There were a total of 20 participants (18 males and 2 females) from 21 to 56 years (*mean=25, std=7.636*), 85% of the test users were programmers. Six of the participants had been students of MCP and used SmartBoards but none of them had ever used the back-end of the system.

5.2.2 Procedure

Initially there was a pilot test which helped to understand where users would have difficulties. Afterwards there were improvements to the information given to the users before the tasks, about how to use the system so they could be more prepared as well as some alterations to the test procedure. Before testing each system users were given an explanation of its functionalities and a sheet of paper with a summary of how the language functions, containing a list of the most important functions and examples. Since the front-end didn't have significant differences between versions, tasks were given with clear indications of which elements they had to manipulate. This helped prevent users from wasting to much time in parts of the tasks that were the same in both systems.

In each test, the user would perform the same eight tasks, with a randomized order, on both systems, referred as systems A (new version) and B (old version). Everyone tested both systems but half of them started with A and the remaining with B to decrease the learning effect. because there are still many commonalities between the two versions which can cause the second system to have faster execution times. Before starting they were given and explanation of how the View editing works and they also had access to a sheet with important information. Afterwards they were asked to perform the following tasks:

1. Add a table, then edit its layout so it has a header row and three columns. Afterwards change the name of the columns to "Name", "Description" and "Number of Levels";

2. Starting with a pre-made table (of the previous task and with added content inside the table), edit the settings of the table row to make the table display all badges;

3. Starting with a pre-made table with the data of all users, columns for their information and the

element in the ”XP” column already filled in. Change the contents of the element in the ”Name” column to make it display the name of the user in that row;

4. Switch the element in the ”Photo” column to an image and then edit it to make it display the photo of the user.

5. Add to the element in the ”Level” column the level of the user;

6. Add to the element in the ”Level” column the name of the level of the user;

7. Edit the element in the ”Ranking” column so it displays the index of the users being looped through plus one, so that it doesn't start at zero;

8. Edit the row setting to change the order of the users being listed, now they should be sorted by XP in descending order.

Each task was timed and if they surpassed five minutes it would be considered a failure. After all the tasks of a system were finished, the users filled in a questionnaire with the System Usability Scale (SUS) and the NASA Task Load Index (NASA-TLX).

5.2.3 Results

The success rate of each task was calculated, which provided some insights into what were the more troubling tasks:

Table 5.2: Success rate of each task per system

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Old	100%	100%	95%	60%	95%	50%	95%	100%
Current	100%	100%	100%	95%	90%	95%	100%	90%

We can observe here that while most tasks had a 90 to 100 percent success rate, there were two with significantly lower results in the old system. Both of these tasks required more complex expressions in the old version.

The time to complete each task was measured. Afterwards, we calculated the mean, median, standard deviation, and confidence interval for each one:

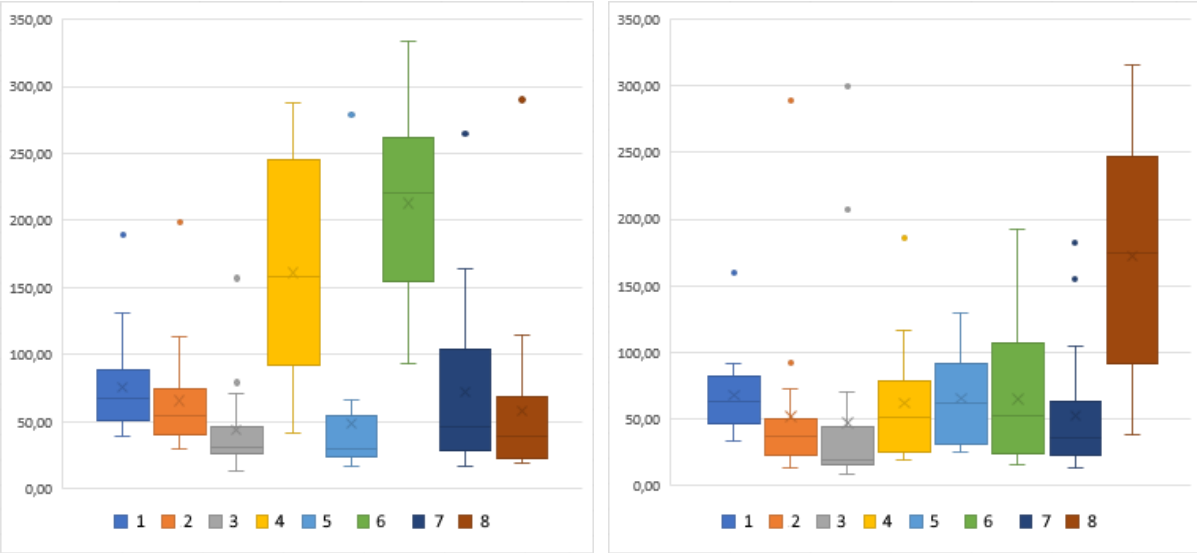
Table 5.3: Time (in seconds) to complete each task on the old system.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Mean	75,60	65,60	43,47	160,83	48,47	212,80	72,37	57,65
Median	67,50	54,50	31,00	157,50	29,00	220,50	46,00	38,50
Std. Deviation	35,308	38,926	33,135	84,938	57,828	72,291	62,933	60,994

This information was then used to create boxplots (Fig. 5.1), where we can clearly see that the most difficult tasks to complete on the old system were task four and six, and on the new version it was task

Table 5.4: Time (in seconds) to complete each task on the current system.

	1	2	3	4	5	6	7	8
Mean	67,60	51,65	47,10	61,89	65,11	64,79	52,15	171,89
Median	63,50	37,50	19,00	51,00	61,50	52,00	35,50	175,00
Std. Deviation	28,719	59,336	73,711	42,659	33,262	48,981	46,200	85,960



(a) Boxplot of the old system

(b) Boxplot of the new system.

Figure 5.1: Boxplots of seconds to complete tasks in each system.

eight. The remaining tasks all have confidence intervals under the 200 seconds line. Subsequently, we performed Shapiro–Wilk test on the times of each of the tasks to check if they had a normal distribution in order to know what comparative test to perform afterwards. While some of the data had a normal distribution, there was not any case where both samples we needed to compare were normal. Thus, the test performed for all tasks was the Wilcoxon signed-rank test.

We performed Wilcoxon signed-rank tests comparing the times of each task on both systems. The null hypothesis was that the median difference between the samples was zero. Tasks one and seven retain the null hypothesis, while the remaining tasks had statistically different results.The first is a task that does not use any of the language and is simply based on using the interface, which is very similar on both versions. The latter is fairly simple to accomplish on both versions. Results displayed in Table 5.5 show that most tasks where there was a statistically significant difference, had faster execution time on the new system. However there were two cases where the opposite happened.

The task with the worst results for the new SmartBoards was task eight (ordering a table of users by XP), where users had to write the expression {**users.getAllUsers.sort(”descending”,”xp.getXP(%item)”)**} which is quite complex, while in the old system they had to select an option from a drop-down menu and

Table 5.5: Results of Wilcoxon signed-rank tests.

Task	Null Hypothesis Decision	Significance	Fastest Task Times
1	Not Rejected	0.332>0.05	—
2	Rejected	0.007<0.05	New SmartBoards
3	Rejected	0.011<0.05	New SmartBoards
4	Rejected	0.003<0.05	New SmartBoards
5	Rejected	0.031<0.05	Old SmartBoards
6	Rejected	0.005<0.05	New SmartBoards
7	Not Rejected	0.083>0.05	—
8	Rejected	0.001<0.05	Old SmartBoards

write a simple expression. The sorting is done with a function to allow more flexibility, since this way the sorting can be done at any moment and in conjunction with other functions if we want it to. However, this can result in bigger difficulty to write expressions, so a possible solution would be to have it both ways, keeping the function but adding an easier way to sort. The other task with a significantly higher time in the new system was task five. This appears to be a result of having a larger expression in this version.

Tasks four and six where the ones with the most significant difference, with longer times in the old version. Besides the discrepancy in the task execution times, they also had the lowest success rate, 60% and 50% respectively. Task six required users to write the expression to get the name of an user's level which in the old system was {**moduleData.xp.levels[** { **$int(%user.data.level)** } **].title**}, while in the new one it was {**xp.getLevel(%item).description**}. The problem in the old SmartBoards was not only that the expression was longer, it also required two steps that were not obvious, using more curly brackets inside the square brackets, and using the **$int** function. If users did not write these details they would get a confusing fatal error message. Additionally, when they finally got the correct expression the square in the expression field, which indicates if the syntax is correct, would incorrectly say the expression was invalid. The other complicated task was adding the users' photos to the leaderboard. In the new platform this could be done by writing {**%item.picture**}, while in the other it required the expression **photos**/{**users[%userkey].username**}**.png**, but often users would not remember they had the variable **%userkey** and ended up using {**%user.id**} instead, which has the complexity of using more curly brackets. Besides having a longer expression, one of the reasons for the difficulty of this task in the old system is that they already had access to a variable with a course user object but it did not have the username parameter because that was in another map of users. Seven users avoided this problem by using a alternative solution which is not completely correct, since it would not work on all cases. They took advantage of the fact that students usernames (e.g. ist181205) typically have the user id in it (e.g. 81205), thus, they only wrote **photos/ist1**{**%user.id**}**.png**. If we penalized the execution times of those who did the alternative solution, the average times of execution would have been even higher. The other tasks that showed significant differences in favor of the new system were number two and three, which both had a simple solution on either system, but the results showed users executed them better in the new version.

Users answered two questionnaires, SUS produces a score from 0 to 100, with 100 being the best score. We used a variant of the NASA-TLX called Raw TLX which includes just the questions, unlike the original assessment tool which also takes into account weighed scales. This results in a value from 0 to 100, with 0 being the best score. The results obtained from these tools are presented in Table 5.6.

Table 5.6: Scores obtained in the questionnaires.

		Old System	New System
SUS	**Mean**	44.25	57.625
	Median	46.255	56.25
	Std. Deviation	14.781	14.678
NASA-TLX	**Mean**	35.417	28.917
	Median	37.5	29.583
	Std. Deviation	11.1	13.707

The data obtained from these questionnaires was also tested to check if they were statistically different. We started by performing a Shapiro-Will test which showed that all the samples came from a normal distribution. Given that information, we then performed a T-Student test for the data of each tool. On the SUS data we got $p=0.001<0.05$, thus we can affirm that the new system has statistically significant higher values than the old system. The values from the NASA-TLX also showed a significant difference, resulting in a better score by the new system with $p=0.018<0.05$. These scores where not necessarily great results because the interface still has a large quantity of usability problems. However both of them were significantly better in the new version than in the old one.

5.2.4 Case Study

The previously mentioned user tests were only able to evaluate features in common between the two systems. However there are many other factors that should be evaluated. Therefore, we had an user test as a case study, with a participant who is an expert of usability and visualization, and who previously taught MCP, so he clearly understand what was the purpose of SmartBoards and could provide important feedback. This user had used the Moodle platform to evaluate students' work on this course. However, even though he taught MCP he had no prior experience with the faculty back-end of the SmartBoards system since that is only usually used one of the professors.

This test included the same tasks, but without being timed and using a think-aloud protocol. After these, there was an additional demonstration of the remaining functionalities of the systems which weren't used in the tasks, and the user was asked to provide opinions on the flaws and features of each system. This way we could get a more in depth comparisons of the system in the view editing tasks from the perspective of someone who can evaluate usability, as well as opinions of the rest of the features of the systems.

The main issues that exist on both versions are with the front-end. Toolbars have small buttons and sometimes it can be confusing to understand which settings one should access to make the desired

changes. Another problem is that it is not clear that after adding a new element to the View, they have to press the *Edit Layout* button again to be able to edit the element. Finally, the settings overlay only has a close button which also saves it instead of having a button for closing and another for saving.

The test user showed appreciation for the features of the new system which did not exist previously, such as creating and deleting pages directly in SmartBoards and the ability to edit and delete templates, because these functionalities allow users to perform important tasks without the need to change the code of the platform. He also recognized the bugs that were fixed such as the settings overlay no longer inadvertently closing. However, he though that the two different ways to use templates seemed confusing, and could possibly become more clear if it was different in terms of graphical user interface design.

He thought that the new expression language has structured in a way that made more sense and it has easier to use, and the events provided flexibility to do more actions. However, he believes that the View editing and expression language would only be used by advanced users, with the remaining ones preferring to use pre-made templates because although the language makes sense it takes time and work to understand it, and users would probably need help from an expert or a tutorial in order to learn to use it.

There was also some feedback related to how some parts could be improved besides the main front-end issues. Whenever an expression field recognizes a syntax error and indicates it by showing a red square it could also specify exactly what is wrong with the expression. New variables can be used from the moment an expression for them is written, but this should have some kind feedback to show the user that the variable has been defined.

The test user thought that this system could be used in other courses, but it could take some work to customized if its faculty did not want to use the defined templates. When asked if he would use SmartBoards as a professor, he said that if the course was gamified, he would prefer to use the new SmartBoards platform.

5.2.5 Discussion

The user tests allowed us to know what were the difficulties which users had when using the system, and in which ways it was better or worse than the old version. Overall, tests results show improvement on the usability of the language, even though there were a couple tasks with longer execution times in the new version. While there is still room for improvement, this new language appears to have a better structure and be easier to understand.

A major part of the developments made could not be tested with these comparative tests, so the case study provided an evaluation of the rest of the system. New features that give users the ability to create and delete pages without having to code, and the ability to edit and delete modules were considered

very useful. Furthermore, the events, which were not tested in the other tests because they functioned too differently, have a greater flexibility by allowing, for example, to go to any page instead of only having a function to go to profile. The system still has a lack of usability with its interface, but the new language and features helped making it more well-structured and versatile.

6

Conclusion

Contents

For this thesis, we developed a scalable and flexible gamification solution for higher learning courses to replace the previous system being used on MCP. We still have not got a GameCourse system that includes all the disparate parts of the system in a cohesive solution, but we are one step closer to reaching it. For now there is still a Python script which collects data from external sources, but one of those sources was moved to SmartBoards as the new QR module, and the text files that were used by the Python script and SmartBoards are now configurable directly in the platform which facilitates the setting up process since it no longer requires editing text files.

SmartBoards had major updates, including switching from a key-value database to a relational MySQL database, defining and implementing a completely new expression language to be used in the View editor, and fixing many bugs. There were also new features added to this platform. Pages and templates can now be freely created, deleted and edited, and Views can use events to go to any page, show templates as tooltips, and toggle the visibility of an element in the View. All these functionalities provide a greater degree of flexibility and versatility compared to the old system, since now there is much more that can be achieved in the platform without having to add more code to the system.

The language and View editing process were evaluated with the user tests, where users were given tasks to perform which resulted in creating a leaderboard. Even though users had some struggles to learn how to use the system, most of them were able to successfully complete the tasks in the new system. Most tasks had positive results favoring the new SmartBoards, but there were a couple tasks with better results in the previous one. Users were also asked to fill a SUS questionnaire which resulted in an average score significantly better than the result of the old system, there was also a Raw NASA-TLX questionnaire which also had a statistically better result in the newer version. Most of the criticism about the system were complaints about the front-end of View editing page, which did not improve much between versions because the focus of this thesis was on the back-end.

The benchmark test showed improvement in load times on the intermediate version of the system. However, the current version still needs optimizations. The case study evaluation showed that the new system is preferable to the previous one, given its improved language which is more structured, and the additional important features that provide more flexibility.

Overall, this system in its current state provides a way to gamify a course, allowing for customization of game elements and Views. Students can have their own personalized learning experience with gamification elements adapted to their learning styles, which may lead to a better gamification experience since users who may not respond as well to specific game elements can now have alternative Views specifically catered to their characteristics. Any professor can create any number of courses on SmartBoards, and they may extend the system if they wish to add new functionalities by creating new modules. This system is a huge step in Learning Systems because it provides a scalable and flexible way to gamify courses with the freedom to customize every single game element, and the power to

adapt the experience according to each student's needs. This will help keeping all students engaged in the course and motivated to perform better.

6.1 Future Work

This system has not yet reached its final stage, which would have SmartBoards become GameCourse without the system being dependent of an external script to gather and process its data. In order to reach that stage we will need a rule system to process course data and replace the Python script. This system should also automatize its updating process so the faculty of MCP can finally stop running scripts regularly in order to update the website, and it should be able to integrate with other platforms besides Moodle and be able to use more authentication methods. Additionally, GameCourse should have a test suite to help with its maintenance, and a caching system to improve the speed of the View rendering process.

Another important part of the future developments is the front-end of the platform. It did not have any major improvements during the present work, and it is one of the main complaints about the system received by test users. The View editing page has toolbars with small buttons which can be difficult to press, and the interaction can be somewhat confusing for new users. Therefore, updating the front-end is one of the main priorities in order to improve the usability of the system.

Finally with the feature of having multiple aspects for a View, we can now proceed with studying how gamification impacts different types of students and develop alternative Views for subsets of students in order to provide a positive pedagogical impact.

Bibliography

[1] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: defining gamification," in *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*. ACM, 2011, pp. 9–15.

[2] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *2014 47th Hawaii international conference on system sciences (HICSS)*. IEEE, 2014, pp. 3025–3034.

[3] B. Leong and Y. Luo, "Application of game mechanics to improve student engagement," in *International Conference on Teaching and Learning in Higher Education*, 2011.

[4] G. Kiryakova, N. Angelova, and L. Yordanova, "Gamification in education." Proceedings of 9th International Balkan Education and Science Conference, 2014.

[5] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "Studying student differentiation in gamified education: A long-term study," *Computers in Human Behavior*, vol. 30, p. 1e36, 2016.

[6] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.

[7] ——, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *American psychologist*, vol. 55, no. 1, p. 68, 2000.

[8] R. Bartle, "Hearts, clubs, diamonds, spades: Players who suit muds," *Journal of MUD research*, vol. 1, no. 1, p. 19, 1996.

[9] L. E. Nacke, C. Bateman, and R. L. Mandryk, "Brainhex: preliminary results from a neurobiological gamer typology survey," in *International Conference on Entertainment Computing*. Springer, 2011, pp. 288–293.

[10] A. Marczewski, *User Types HEXAD*, 10 2015.

[11] G. F. Tondello, R. R. Wehbe, L. Diamond, M. Busch, A. Marczewski, and L. E. Nacke, "The gamification user types hexad scale," in *Proceedings of the 2016 annual symposium on computer-human interaction in play*. ACM, 2016, pp. 229–243.

[12] K. Werbach and D. Hunter, *For the win: How game thinking can revolutionize your business*. Wharton Digital Press, 2012.

[13] A. F. Aparicio, F. L. G. Vela, J. L. G. Sánchez, and J. L. I. Montes, "Analysis and application of gamification," in *Proceedings of the 13th International Conference on Interacción Persona-Ordenador*. ACM, 2012, p. 17.

[14] E. D. Mekler, F. Brühlmann, K. Opwis, and A. N. Tuch, "Do points, levels and leaderboards harm intrinsic motivation?: an empirical analysis of common gamification elements," in *Proceedings of the First International Conference on gameful design, research, and applications*. ACM, 2013, pp. 66–73.

[15] W. H.-Y. Huang and D. Soman, "Gamification of education," *Research Report Series: Behavioural Economics in Action, Rotman School of Management, University of Toronto*, 2013.

[16] D. Goshevski, J. Veljanoska, and T. Hatziapostolou, "A review of gamification platforms for higher education," in *Proceedings of the 8th Balkan Conference in Informatics*. ACM, 2017, p. 28.

[17] B. B. Morrison and B. DiSalvo, "Khan academy gamifies computer science," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 39–44.

[18] H. Coates, R. James, and G. Baldwin, "A critical examination of the effects of learning management systems on university teaching and learning," *Tertiary education and management*, vol. 11, pp. 19–36, 2005.

[19] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "Engaging engineering students with gamification," in *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2013 5th International Conference on*. IEEE, 2013, pp. 1–8.

[20] ——, "Improving participation and learning with gamification," in *Proceedings of the First International Conference on gameful design, research, and applications*. ACM, 2013, pp. 10–17.

[21] ——, "Gamification for smarter learning: tales from the trenches," *Smart Learning Environments*, vol. 2, no. 1, p. 10, 2015.

[22] ——, "Early prediction of student profiles based on performance and gaming preferences," *IEEE Transactions on Learning Technologies*, vol. 9, pp. 1–1, 01 2016.

[23] K. Liang, Y. Zhang, Y. He, Y. Zhou, W. Tan, and X. Li, “Online behavior analysis-based student profile for intelligent e-learning,” *Journal of Electrical and Computer Engineering*, vol. 2017, 2017.

[24] H. Bydžovská, “A comparative analysis of techniques for predicting student performance,” in *Proceedings of the 9th International Conference on Educational Data Mining*, vol. 2016, 2016.

[25] C. Romero and S. Ventura, “Educational data mining: a review of the state of the art,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 6, pp. 601–618, 2010.

[26] A. Silva and C. Antunes, “Mining multi-dimensional patterns for student modelling.” in *EDM*, 2014, pp. 393–394.

[27] A. C. R. do Vale, S. C. Madeira, and C. Antunes, “Mining coherent evolution patterns in education through biclustering [abstract and poster],” in *7th International Conference on Educational Data Mining*. International Educational Data Mining Society (IEDMS), Jul. 2014, pp. 391–392.

[28] M. Chi, D. Schwartz, K. P. Blair, and D. B. Chin, “Choice-based assessment: Can choices made in digital games predict 6th-grade students’ math test scores?” in *Educational Data Mining 2014*. Citeseer, 2014.

[29] C. Antunes, *Anticipating student’s failure as soon as possible*. Chapman & Hall/CRC Press, New York, NY, 2010.

[30] J. McBroom, B. Jeffries, I. Koprinska, and K. Yacef, “Mining behaviors of students in autograding submission system logs.” in *EDM*, 2016, pp. 159–166.

[31] S. J. Lee, Y.-E. Liu, and Z. Popovic, “Learning individual behavior in an educational game: a data-driven approach,” in *Educational Data Mining 2014*, 2014.

[32] S. Klingler, T. Käser, B. Solenthaler, and M. Gross, “Temporally coherent clustering of student data.” *International Educational Data Mining Society*, 2016.

[33] J. Amaral, “Gamecourse,” 2013.

[34] A. Baltazar, “Smartboards,” 2016.

A

Appendix A

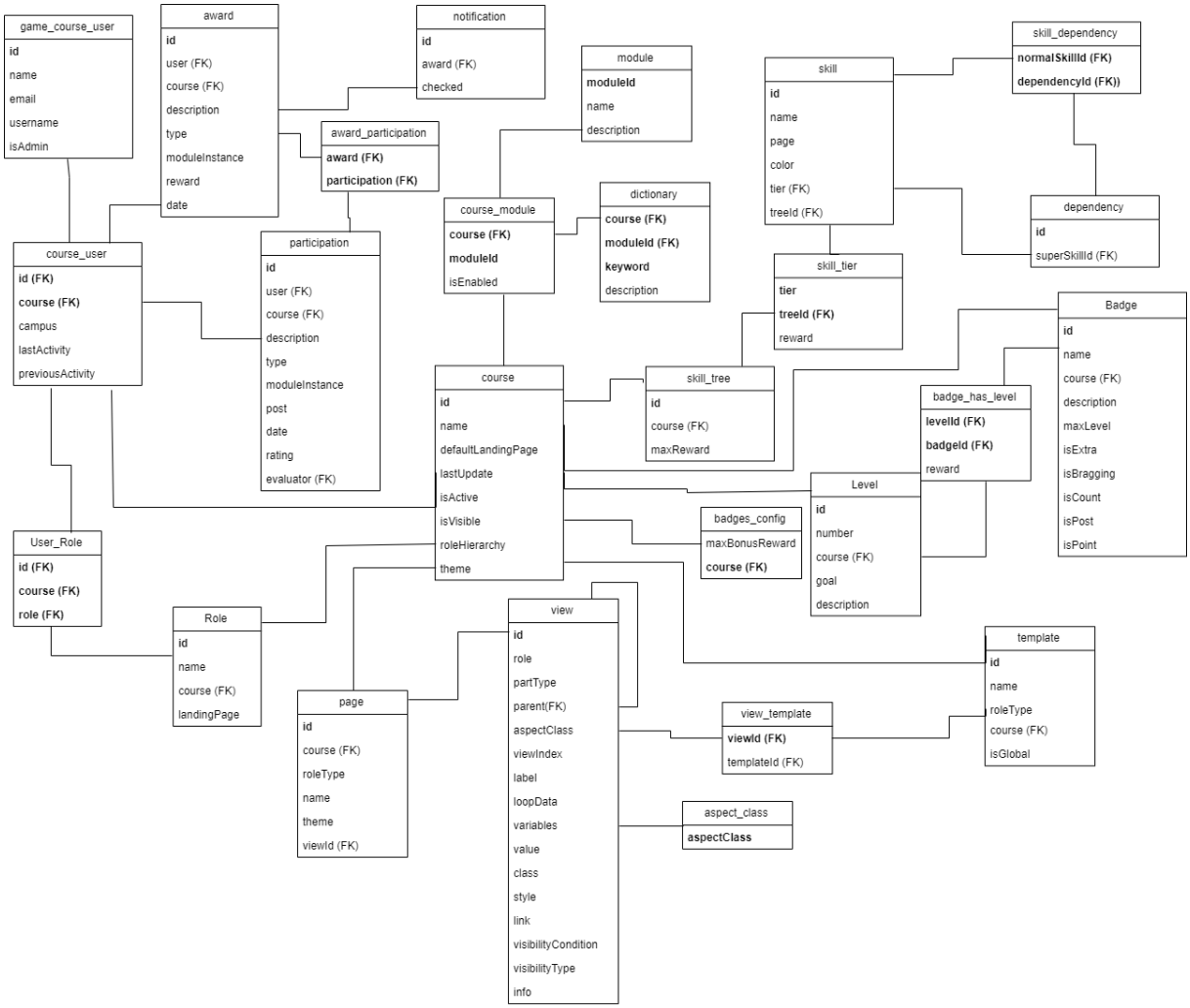


Figure A.1: Complete Schema of GameCourse database with Skills and Badges enabled.

B

Technical Report

B.1 Installing and Updating SmartBoards

To setup SmartBoards, use the following steps:

1. Set up a Fénix application to allow students to log in using their IST credentials;
2. Update the *config* file with database and Fénix information;
3. If there is a file called *setup.done*, delete it;
4. Go to the SmartBoards page to create a course;
5. In the course settings, enable the desired modules then go through all configuration pages (e.g. Students, Teachers) and update the data lists. In case we do not want to configure badges, skills, or levels from scratch there will be configuration examples on *legacy_data/defaultData*;
6. Run the Python script;
7. On the settings page of the course, click on *Load Legacy*;
8. Insert Fénix cookie information from in the JSESSIONID and BACKENDID fields, and press *Download Photos*;
9. Now the Views of pages can be configured.

To update a course in SmartBoards, use the following steps:

1. Run the Python script;
2. In the course settings page, click on *Load Legacy google-api-php-client* folder.

Additionally, if we want to compile *.less* files we should install Less, and to change the compiler of the expression language (*.jison* file), Jison should be installed, and in *generateParsers.sh* the correct path should be set in JISON_PHP. To be able to use Google authenctication, it is necessary to create credentials on Google API Console and replace them in the *google-api-php-client* folder.

To create a new course, go to GameCourse settings (do not confuse with the settings of a specific course) and in the Courses tab press *Create New*. Now you can choose between creating an empty course or one similar to an existing course.

B.2 Editing Views

View of Pages and Templates can be edited by selecting them on the side menu of the settings page and then selecting the aspect we want to edit. On the View editing page every element has a toolbar.

Elements can be added by pressing the *Edit Layout* button, which has the image of a brick wall. After pressing this button (Fig. B.1) we can add new parts to the View, we may also add templates by copy (using the *Add New Part* drop-down) or by reference (using the second drop-down). Next, in order to edit the newly added elements, we need to press the *Edit Layout* button again, and now we can configure these Views

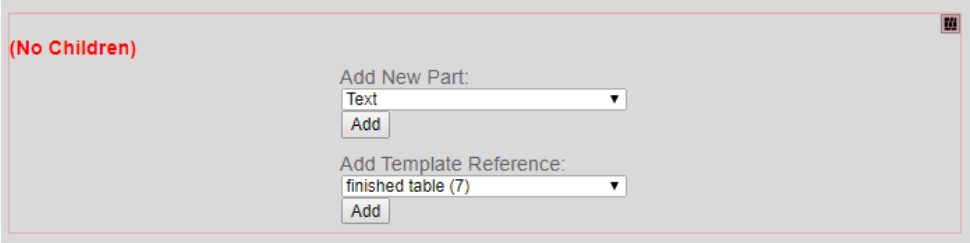


Figure B.1: Editing page of an Empty View, after pressing *Edit Layout*.

Views have individual configurations where we can define their parameters. As an example, to create a list of all users we could add a text View and edit it to have all users on its Data and the ID of a user in its Content (Fig. B.2). It would produce the result in Figure B.3.

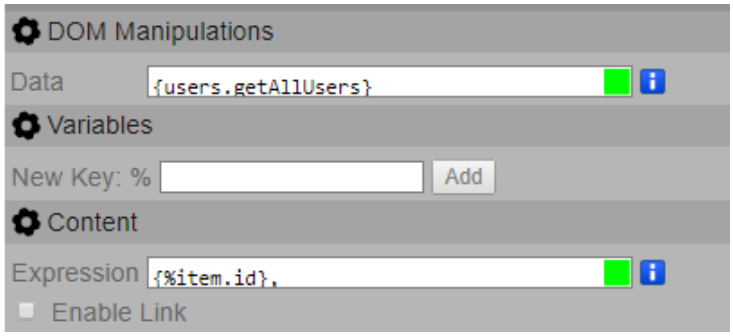


Figure B.2: Configuration of a View that displays IDs of all users.

Figure B.3: View that displays ids of all users.

B.3 Module Installation

Creating modules is the way to extend the system and add new functionalities.

To create a module called example, we start by creating a directory inside the *modules* folder with the name *example*. Inside it we should have a file *module.example.php* (Fig. B.4).

```php
<?php
use GameCourse\Module;
use GameCourse\ModuleLoader;

class Example extends Module {
}
ModuleLoader::registerModule(array(
    'id' => 'example',
    'name' => 'Example',
    'version' => '0.1',
    'factory' => function() {
        return new Overview();
    }
));
```

Figure B.4: *module.example.php - Empty Example Module*

This code, creates a PHP class for the module and registers it with the module loader, which will load it whenever it is required. To initialize something when the module initializes the function *init()* is used. It allows the module to use other features of the SmartBoards core or setup additional things required for the module. It could be used for example to add a template (Fig B.5).

```php
class Example extends Module {
    public function init() {
        $viewsModule = $this->getParent()->getModule('views');
        if (!$viewsModule->templateExists("Example Template")){
            $viewsModule->setTemplate("Example Template", file_get_contents(__DIR__ . '/example.txt'));
        }
    }
}
```

Figure B.5: Adding a template in init() function of *module.example.php*.

In this case, we use functions provided by the views module to set the template located in *modules/example/example.txt*, which will get the name ”Example Template”.

If we wish to add behaviour from Javascipt or CSS files, they shoud be added inside the function *setupResources()* (Fig. B.6) where we can specify which folders or files we want to use.

```php
public function setupResources() {
    parent::addResources('js/');
    parent::addResources('css/');
}
```

Figure B.6: *module.example.php* - setting up resources.

Another important functionality is registering new functions in the expression language (Fig. B.7). To do this we need to get the viewHandler from the views module and then call the function *registerFunction*

which received the name of the library, name of function and its behaviour.

```
public function init() {
    $viewsModule = $this->getParent()->getModule('views');
    $viewHandler = $viewsModule->getViewHandler();
    $viewHandler->registerFunction('exampleLib','exampleFun', function($arg) {
        return new ValueNode($arg);
    });
```

Figure B.7: Defining an expression language function in *module.example.php*.

In the modules, just like in any other part of the code, the databased can be accessed by calling functions over `Core::$systemDB`. There are many functions available such as *select*, *selectMultiple*, and *update*, which are defined in the *SQLDB.php* class. To be able to acess the Core, the file must have the expression *use GameCourse/Core;*. For example, to select the user with ID = 1111 we would write `Core::$systemDB->select("game_course_user", ["id"=>1111]);`

B.4 Files of the System

This system has a folder with classes that has important files such as *Course.php*, *CourseUser.php* which have behaviour for their respective objects, and also *Core.php* containing general functionalities of the system. Another important class is *SQLDB.php* which, as mentioned before, has all the functions used to interact with the database.

All behavior related with settings page (except View editing) is defined in *js/settings.js*, *js/configurations.js*, and *classes/GameCourse/Settings.php*.

The *loadLegacy.php* script is responsible for updating the database after running the Python script. One of the most important files is *info.php* which has API functions that can be called on Javascript files. There are more API functions (related with iew behaviour) defined in the *module.Views.php* file inside its module folder.

Each module has it own folder with all its behaviour which in most cases is fairly simple. The exception is the Views module which has the *module.Views.php* and *ViewHandler.php* files containing all behaviour related with parsing processing and rendering views. To render a View, the following steps will happen:

- The View module receives a request to render a view;
- We verify if the user has permission to access the page, calculate View parameters, and call the function *handle* from ViewHandler;
- The ViewHandler will choose what aspect should be displayed;

- View and its children Views are selected from the database;
- All contents of the View are parsed;
- All contents of the View are processed, which is when the functions of the expression language may be called;
- The contents of the View are returned.

The other complex process is saving Views, which will call the function *updateViewAndChildren* responsible for updating every single View from the array given to the tables of the database, this update process may add, delete and update Views.