SmartBoards

André Maurício Baltazar

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. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. António Paulo Teles de Menezes Correia Leitão

October 2016

Acknowledgments

First, I would like to thank my family that always supported me in every decision, and allowed me to be the person I am today. Second, I would like to thank professor Daniel Gonçalves for allowing me to develop this awesome project, and helping me through every step necessary to complete it. It is also very important to mention my friends, which have supported me, and gave suggestions about the project. To all of you, a big thank you.

Abstract

Student motivation is important to excel in school. However, it is sometimes hard to achieve this motivation to study for a course. A technique being increasingly used to try to increase the motivation is called Gamification. This technique applies game elements in non-game contexts to harness the motivational power of games. Gamification is used by the course of Multimedia Content Production (PCM) at Instituto Superior Técnico (IST) to try to engage students. A study suggests, that the students perceive the course as more motivating than the others. Although it is perceived this way, some results suggest that applying gamification can also have negative influence on people, because of perceived competition or other negative aspects of gamification. To try to minimize this influence we developed web application capable of presenting different content to different students. This web application replaces an old system, which generated static pages using a script, with a brand new system, which is modular, configurable and allows to change the presentation of each page easily. Using the tools available in the new system, it is possible to create a different view to be presented to different students, catering to their learning and gaming profiles, and thus presenting each one with a custom-tailored experience, more motivating and engaging. To evaluate the usability of this functionality user tests were conducted. The result shows that the users liked the way the system worked, and in the System Usability Scale test the score was good with an average score of 81.

Keywords: gamification, education, motivation, view differentiation, web application

Resumo

A motivação dos alunos é importante para permitir-lhes que se destaquem na escola. No entanto, por vezes é difícil de conseguir motivação para estudar. Uma técnica cada vez mais utilizada para tentar aumentar a motivação é chamada Gamificação. Esta técnica utiliza elementos de jogo em contextos não-jogo para aproveitar o poder motivação dos jogos. Esta Gamificação é usada pela disciplina de Produção de Conteúdos Multimédia (PCM) no Instituto Superior Técnico (IST). Um estudo sugere que os alunos entendem a disciplina como mais motivadora do que as outras, mas embora ela seja percebida desta forma, alguns resultados sugerem que a aplicação de gamificação pode ter uma influência negativa sobre as pessoas, devido à perceção de competição ou outros aspectos negativos da gamificação. Para tentar minimizar esta influência desenvolvemos uma aplicação web capaz de apresentar conteúdo diferente para diferentes alunos. Esta aplicação web substitui um sistema antigo, que gerava páginas estáticas usando um script, por um sistema novo, que é modular, configurável e permite alterar a apresentação de cada página com facilidade. Usando as ferramentas disponíveis no novo sistema, é possível criar uma vista diferente a ser apresentada a diferentes alunos, atendendo aos seus perfis de aprendizagem e de jogo, e apresentando assim a cada um, uma experiência sob medida, mais motivadora e envolvente. Para avaliar a usabilidade desta funcionalidade foram realizados testes com utilizadores, que mostraram que estes gostaram da forma como o sistema funcionava. No teste de System Usability Scale a pontuação foi positiva, com uma pontuação média de 81.

Palavras-Chave: gamificação, educação, motivação, diferenciação de vistas, aplicação web

Contents

List of Tables

List of Figures

Acronyms

IST Instituto Superior Técnico. 3, 8, 19, 43

PCM Produção de Conteúdos Multimédia. 3–5, 8, 9, 14, 15, 17, 19, 20, 23, 24, 34, 38, 39, 49–51, 53, 54, 61

PHP PHP: Hypertext Preprocessor. 23, 26, 29, 31, 40, 43–45, 50, 53

SUS System Usability Scale. 55, 57

XP Experience Points. 17, 37, 45, 49

Chapter 1

Introduction

Engagement and motivation are important to make people excel, be it at their job or in their areas of study. It is required for humans to take action and work better. However in education it is hard sometimes for students to engage with the subjects they are studying, because they lack motivation. There are two kinds of motivation. Intrinsic motivation is a behavior driven by the predisposition to accomplish the tasks while extrinsic motivation is driven by some sort of reward. With the introduction of a way for students to engage more with the courses it can create more motivation and interest to learn the subject. One of the ways to create this engagement is by applying Gamification, the use of game elements in non-game contexts [4]. Games can provide this engagement as they are one of the oldest forms of human interaction[1] and can drive the behavior of players and motivate people because they can introduce a challenge that is on par with the players' abilities. Video games are sometimes used in education and help improve critical thinking and problem-solving skills. However, gamification only uses the game elements present in the games, such as leaderboards, points and badges, and adapts them to the context in order to make users engage in specific behaviors like competition or cooperation. These game elements are used in many places from work to educational environments, where there are some good examples of gamification in online education, like Khan Academy[2] which uses points and badges as a reward for learning. This kind of reward can create extrinsic motivation on people that already wanted to learn (those who have intrinsic motivation) and further engage them to continue learning and have an overall better experience.

Although gamification can engage some students, studies suggest that it can also have negative influences on people. This problem occurs because even though the game elements are intended to motivate people, they are perceived as bad for other people. This is also due to perceived competition of certain game elements, like Leaderboards, which rank the users, and make them compete for a good spot in the leaderboard. The motivation for the development of this thesis was to create a system that can be configurable to solve problems, like these negative effects. It can also improve the motivation of the students allowing for a better engagement in a gamified course, like the course of Produção de Conteúdos Multimédia (PCM)(Multimedia Content Production) at Instituto Superior Técnico (IST). This course uses gamification to try to engage students, and allows them to make their own way through the course, with the possibility of focusing more on some subjects than others, and is perceived by students as more motivating than other courses[1]. In the most recent years, it has been using the game elements of experience points, badges, skills, leaderboard, profile pages, and quest, which can be displayed in many different ways, and the way one student perceives the element could be different from another student.

[1] https://en.wikipedia.org/wiki/History_of_games
[2] https://www.khanacademy.org/

Even though, in general, the students liked the platform which was used in the years prior to the 2015/2016 semester, it had a few problems. It was developed and improved over the years to enable the gamification of the course, but maintaining and adding new features was difficult. It displays the information of leaderboards, personal profiles, badges, etc., but it is not structured and the pages are static. A Python script is used to generate them, and adding new features to it is difficult and does not allow the option to display different information to different students, because the pages are statically generated. There is also a lack of a back-end where the professors can monitor students' performance and change any setting they need about the system with ease, instead of having to go and edit the script.

Our main objective was **to create a new dynamic web solution to replace the static pages with an interactive site that allows differentiated forms of presentation to different students** and support the gamified experience of PCM. Allowing for these differentiated forms of presentation of the leaderboard, personal profiles, etc. is important because this information can be displayed in a way that is the best for the students to try to keep them motivated. It also allows students that do not normally engage the course to have a better experience by, for example, trying to decrease perceived competition while displaying the leaderboard or displaying the progress towards getting a badge in a different way. This new solution also allows for a better interaction and visual presentation of the different elements of the site, and enables the creation of a back-end for the professors to see how the course is doing and configure some aspects of the platform, like how the information is displayed to a certain student.

In order to make it simpler to implement this solution, it was stated that the system was not required to process all the information of the course. This task was delegated to the old script, which is run whenever necessary to update the information present in the new system, making it easier to focus on the main objective of the project with is the presentation rather than the manipulation of the data. This simplification allowed us to have a working version of the system to be used by the course of PCM in the second semester of 2015/2016. To complete our objective of displaying the information in the best way possible for each student it was necessary to study previous work related to the game elements present in the gamification of PCM. These studies allowed us to get an idea of the capabilities necessary for the system, based on the different types of game elements present in the examples of gamification studied. Using these capabilities allows us to define what is necessary to be easily edited using the interface, so that those changes can be done by a professor in the settings page of the system, instead of needing to change the code of the system.

Chapter 2

Related Work

In this chapter previous research done on gamification is presented, with a focus on education and universities, and the game elements that are used in the gamification of PCM.

2.1 Gamification

Gamification is used to try engage the people with their tasks by using rewards and the sense of achievement/completion to motivate the users. However the introduction of this gamified layer can sometimes create problems of motivation and allow users to have a sub-optimal performance when completing their tasks. Nowadays, gamification is used in many different contexts like fitness/health, education and training or to improve employee productivity in the work space. One of the most widely used example of gamification is Foursquare[1], which rewards users for visiting specific locations. Other services use gamification to create incentive to help others with their questions, e.g. Stack Overflow[2]. One other example of gamification in education online is Codecademy[3] which uses points and badges to track student progress. The use of these systems in education, try to engage students and make the experience of learning more fun.

According to Deterding *et al.*[4] the first documented use of the term dates back to 2008 and they define it as "use of game design elements in non-game contexts". They reckon that if games are designed for the purpose of entertainment then using game elements in non-game contexts should make them more enjoyable and engaging as well.

Deterding[5] presents three pieces, written by different authors, that give an insight into the state of thinking about potentials and limitations of gamification. Judd Antin said in his piece that "observers were amazed by how much internet users would do for free", however with gamification, it is shown that participants rarely work for free and have rewards like self-efficacy, group identification and social approval. He stated that "gamification has a long way to go to achieve its potential" and hopes that research is done to help "produce systems that use social psychological processes well by taking into account contexts, meanings and individual differences". Elizabeth Lawley expresses, in the second piece, that implementing a gamification system can fail to engage players and damage the existing interest or engagement with the service. She created a system of achievements to try to improve students' retention rates by having students make social connections with other students and staff. The system worked well and she stated that the activity itself was the reward for the student, not the achievements. In the final piece Rajat Paharia talks about intrinsic motivation, the necessity of an existing intrinsic value in

[1] https://foursquare.com/
[2] http://stackoverflow.com/
[3] https://www.codecademy.com/

the activity being gamified and that gamification should deepen the engagement and desire to participate. Understanding the users and their needs is one of the requirements for gamification. Deterding concludes that gamification should be taken seriously and the needs of the users should be taken into account, the deployment of a gamified system should not just be implementing common game features and gamification should be a "means to identify and facilitate the motivation behind desired activities, using game design as one guiding lens among many".

When designing a gamified system we should make sure it will not harm the user in any way. Knaving and Björk[6] suggest how to approach gamification and avoid common issues. Referring to previous work of others, they explain the difference between gaming and playing. Gaming is rule bound and goal oriented while playing is a open-ended activity with the possibility of exploration. Following the concept of "flow", the term coined by Csíkszentmihályi, which is defined as "the optimal experience of an activity, that is reached when goals are clear, feedback is immediate, and there is a balance between challenges and skills", they suggest that "formalizing gaming from playing activities can help increase the chance of people having optimal experiences". Applying gamification to a non-game context generally requires the designer to encourage some aspect of the activity in order to simplify the experience, but this simplification can encourage suboptimal behavior or obscure the activity, steering the user from the main activity because they are too focused on the gamification. Having a gamification layer can harm the intrinsic motivation of the activity if the gamification acts as an extrinsic motivation for the activity, in the case the user is not willing to engage an activity with few or no advantages in real life. Another problem is that users may try to "game the system" by manipulating flaws without knowing is that is the desired outcome of the activity. They suggest that the gamification layer should be opt-in or invisible and should not spam unwilling users with information. The gamification layer needs to be motivating and make the user feel competent and autonomous allowing the possibility of exploration allowing "to find and develop intrinsic motivations related to the activity".

There are some questions regarding the effects of gamification. Hamari *et al.*[7] tackles one of these questions by researching previous work to respond do the question "Does Gamification Work?". They found that the most common tested game elements were the Points, Leaderboards and Achievements/Badges. The majority of the studies they reviewed revealed that indeed gamification does produce positive effects. Yet, some studies showed that these effects could be just caused by a novelty effect, and that removing the gamification experience while those users are still engaged could have negative effects, due to the loss of points and badges. The studies in education/learning context had a positive outcome in the presence of gamification, but still there are some effects that need to be taken into account, like increased competition. Responding to the initial question of "Does Gamification Work" they conclude that indeed it works, but there are some problems, for example, the limited time frame of those studies.

Meder and Jain[8] try to tackle the gamification design problem in order to maximize an overall goal. They reckon that since the players respond to game elements in a different way, the use of game elements to cover all players, could have have a negative influence. It is suggested the creation of a generic model based on the user interaction that could help assign a game design element to each user in order to maximize their expected contribution to achieve some goal. This could be done for example by collecting information on the user interaction with the web client interface provided to consult the leaderboard. They suggest that players can be segmented into customer groups, with players that have the same needs and wants, and to each of these groups is assigned an appropriate design element. However, they leave the issue of the change of the user preference open for future research.

In order to investigate the "demographic differences in perceived benefits from gamification" Koivisto and Hamani[9] gathered data via an online survey from users of Fitocracy[4], a service that uses gamifi-

[4]https://www.fitocracy.com/

cation applied to exercise and healthy habits. This service uses badges/achievements, points and levels which are awarded for completing exercises. The study gathered 195 valid responses and results show that "women reported perceiving more social benefits from gamification" and the perceived usefulness and enjoyment of the service diminishes with time, which can be attributed to novelty effects. These novelty effects are stronger with younger users which "is consistent with the general belief that younger people, while being more susceptible to playful interactions, might also get bored more quickly than more mature users". The age of the user "does not affect most of the benefits significantly directly" with only the more mature users having more difficulty using the service (ease of use). They acknowledge that the "data is likely to affect the results as the users responding are most probably actively engaged with the service" since the survey was conducted online and the respondents "likely differ in cultural background, which could affect perceptions of the various elements examined in the study".

2.2 Gamification in Learning

Gamification can be a powerful concept to apply to education and try to motivate learning. Domínguez *et al.*[10] designed a plugin for a e-learning platform and collected data on a university course using that platform. The main objective was to "increase student motivation towards completing optional exercises through the use of rewards and competition mechanisms". The gamified system included a leaderboard to rank the students according the tasks performed and they would accept the completion of the tasks as correct automatically giving the player immediate feedback. Their findings are that students engaged in a gamified experience got higher scores in practical assignments and overall score while performing worse in written assignments. They said that some students did not find it fun to compete with their classmates for a rank in the leaderboard and they suggest the introduction of cooperative and social mechanisms in order to address those issues.

Helms *et al.*[11] applied gamification to improve traditional training in Dutch Railways. They linked the game elements used in gamification with the learning effects that were stated in other research papers and developed a method for designing gamified training programs. Its application at the Dutch Railways concluded that it can increase the intrinsic motivation and engagement of the employees and that designing a gamified training program is not simply adding a leaderboard and points to an existing program. It is noted that the designers should also be aware of the possible side effects like the demotivation of employees due to being ranked low on a leaderboard.

2.3 Gamification on Universities

Indiana University[5][6] professor Lee Sheldon also used gamification in 2010. He taught a Game Design course and applied game elements to the course and says that the "students performed a full letter grade better in the course than students had under the traditional approach". He included game elements like experience points instead of grades and also chance games, like the roll of a dice to decide presentation schedules.

In the University of Michigan, Professor Cliff Lampe[7][8] uses gamification to better engage his students with the course. He goes beyond having just game elements like experience points and quests, engaging the students via Live Action Role Play and Guild Quests. Students are allowed more freedom than in a normal course, with the option to "skip tests and focus on more "artistic" based assessments such as

[5]https://gamingtheclassroom.wordpress.com/syllabus/
[6]http://chronicle.com/blogs/wiredcampus/at-indiana-u-a-class-on-game-design-has-students-playing-to-win/21981
[7]http://games.slashdot.org/story/12/12/10/1827251/professorcliff-lampe-talks-about-gamification-in-academia-video
[8]http://www.gamification.co/2013/02/08/the-gamification-of-college-lectures-at-the-university-of-michigan/

presenting an information graphic of a particular topic". Even though he has no data to back it up, he thinks that the course has a positive impact, because students that participated in this gamified course come to him years later and still remember some of the fun moments of the experience and the content associated with it.

O'Donovan *et al.* [12] applied gamification to a Computer Science course focused on 2D game design and development in order to judge its effectiveness of gamification. Their goal was to"increase student engagement and motivation", "improve lecture attendance and in-class participation" and "enhance content understanding and problem solving skills". They used the Brainhex[9] survey[10] to identify the top three classes of the students, which turned out to be Mastermind, Conqueror and Seeker. From these results they selected the appropriate game elements to include in their gamified course. The selected game elements were badges, progress bars, leaderboards, a storyline and a visual (a background image on the website). In the student questionnaire, the students responded that they think that gamification improved their understanding and engagement, however, they were not certain that it improved their overall grade. Students also ranked the game elements according to their preference and results show that leaderboards and Steam points (points that could be used to redeem quiz do-overs, puzzle hints, class rewards and assignment extensions) were ranked with high preference and ranks, progress bars, end prize and badges ranked low on their preference. They found out that both student grades and class attendance increased compared to the previous year and other courses in the same year, respectively.

At IST, gamification is applied in PCM course. Barata *et al.* [1] discuss the results of gamification on this course and how it can improve the students engagement. They collected data for five years, with the most recent two being in a gamified context. These data contain attendance to lectures, posts and downloads of support material on Moodle[11], and grades. The course uses experience points, levels, a leaderboard, challenges and badges as its game elements and the final grades are assigned to students based on the level they can reach. Results show that gamification did not have an effect on the attendance but had a significant impact in mean posts per student, which increased 4 to 6 times in the first year (of the gamification experiment) and 66% more the next one - see figure 2.1. There was also a 1.5 to 3 times increase of the number of downloads of support material. The final grades were significantly different with an increased mean and minimum of the final grade in the second year of gamification.

In an other study, Barata *et al.*[13] used the data from their previous work and classified the students into different categories using accumulated experience points as the measure of progress. Their analysis allowed to identify three types of students: Achievers, Disheartened and Underachievers. The Achievers were the students with the best grades, attendance and participation. Their mean experience points per day was always ahead of the other two types. Students in the Disheartened type had average-to-low grades, closer to those of the Underachievers. They had an average participation on Moodle and were not proactive. Even though their grades ended up closer to the Underachievers, in the first 6 weeks they where competing with the Achievers. The Underachievers had the lowest grades, performed poorly for most achievements and seem to have taken advantage of easier opportunities to grab experience points. It is suggested that "these students actually enjoyed the course, but it was still just a course". In order to improve students' experience, it was suggested to implement a achievement tree to allow the Achievers to choose they own path and be in control of their learning experience. The Disheartened suggested the implementation of achievements to promote collaboration. The Underachievers could also benefit from these additions but they just seem to lack particular interest in the course. It is also referred that

[9] http://blog.brainhex.com/
[10] http://survey.ihobo.com/BrainHex/
[11] https://moodle.org/

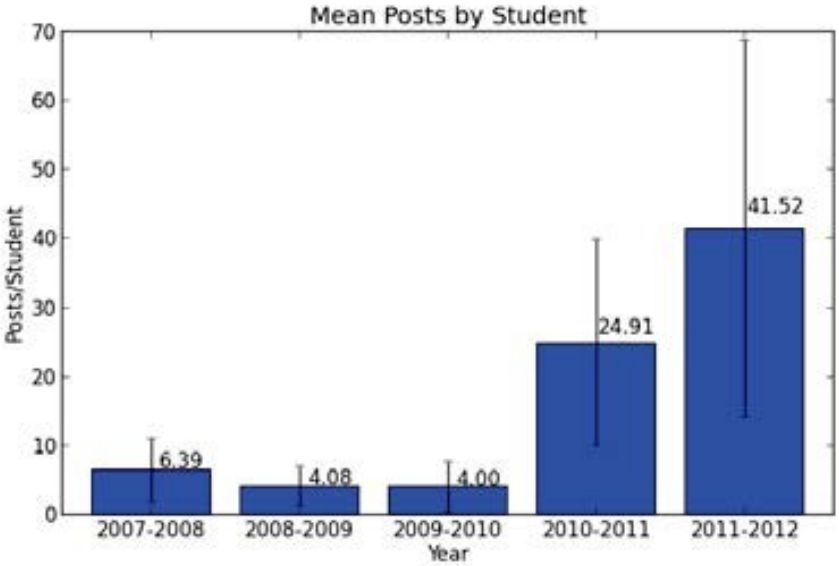


Figure 2.1: Mean posts by student in PCM course. (taken from Barata *et al.* [1])

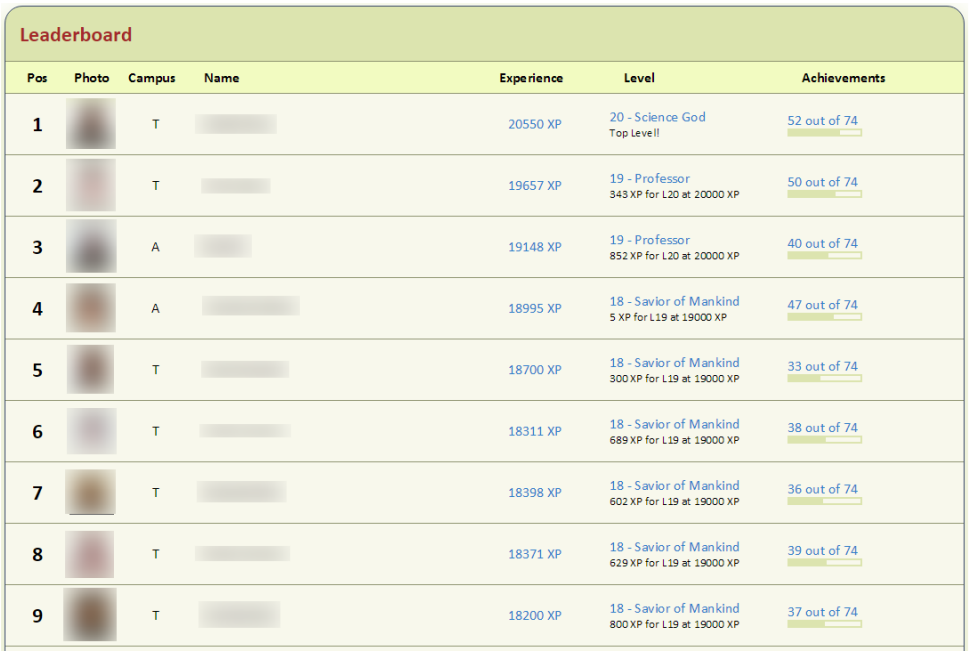
the disinterest by the Underachievers and Disheartened, after 6 weeks, could be due to work for other courses.

2.4 Game Elements - Leaderboards, Points & Badges

There are many kinds of game elements that can be applied to gamification. This project will focus mainly on Leaderboards, Points and Badges. Points and Badges are the rewards for completing tasks, while leaderboards are used to display the ranking of person. Leaderboards can be done in many different ways, figure 2.2 shows the gamified PCM course leaderboard where all the students are in the same leaderboard. Some games like Rocket League[12] have ways to restrict the players on the leaderboards to friends only instead of showing all players in the leaderboards - see figure 2.3a. This game also implements a top ranking leaderboard (as seen in figure 2.3b), which do not display the players own position if they are out of the top, and a league system that places players in different leagues according to their points, which can be seen as a level to reach. Other games reveal the global rank of the player and show only people in a certain range of ranks, near the players rank, for example Miniclip[13], figure 2.4. These different types of leaderboards can have different kinds of impact on people and is important to study their effects to minimize the negative effects that they can have.

Montola *et al.* [14] implemented an achievement system in a photo sharing platform “in order to experiment playfulness outside the game domain”. Other of the aims of their project was “to study the potential of achievements in teaching users to utilize the various features of the service’ and to “encourage users to add new content to the service during the trial”. Users were rewarded with achievement points for sharing content. The study was conducted for eight weeks with 20 people. Results show that the users where indifferent about the achievement system. Those who appreciated the system did not find it essential and those who disliked it did not strongly oppose it. However the authors note that using the achievement system helped users to learn the features of the service. They say that “users need to be notified immediately when they gain an achievement, to remind them of the existence of the

[12]http://rocketleague.psyonix.com/
[13]http://www.miniclip.com/



Leaderboard

Pos	Photo	Campus	Name	Experience	Level	Achievements
1		T		20550 XP	20 - Science God Top Level!	52 out of 74
2		T		19657 XP	19 - Professor 343 XP for L20 at 20000 XP	50 out of 74
3		A		19148 XP	19 - Professor 852 XP for L20 at 20000 XP	40 out of 74
4		A		18995 XP	18 - Savior of Mankind 5 XP for L19 at 19000 XP	47 out of 74
5		T		18700 XP	18 - Savior of Mankind 300 XP for L19 at 19000 XP	33 out of 74
6		T		18311 XP	18 - Savior of Mankind 689 XP for L19 at 19000 XP	38 out of 74
7		T		18398 XP	18 - Savior of Mankind 602 XP for L19 at 19000 XP	36 out of 74
8		T		18371 XP	18 - Savior of Mankind 629 XP for L19 at 19000 XP	39 out of 74
9		T		18200 XP	18 - Savior of Mankind 800 XP for L19 at 19000 XP	37 out of 74

Figure 2.2: The PCM course leaderboard.

achievements, to reward them on the spot, and to arouse their curiosity towards achievements". It is argued that achievements add value to the service, but should allow users to opt out if they want.

A study by Antin and Churchill[15] aimed to understand how or why badges are valuable and useful in a social media context. They argue that the badges have several social functions and present the "five [primary] social psychological functions for badges in social media context: goal setting, instruction, reputation, status/affirmation and group identification". Badges challenge the users to achieve goals and are "known to be an effective motivation" and for some users the "fun and interest of goal seeking is often the primary reward itself". The authors suggest that "goal setting is most effective when users can see their progress towards the goal". The function of "instruction" is identified, because badges show "what types of activity are possible within a given system" providing help to new users and allowing to diversify their participation, while also "exemplifying the types of activities and interactions that are highly valued". The reputation function of badges is their ability to show the interests and engagement level of the users that is linked with the status function of badges which advertise the user's accomplishments and can be motivating. Having the ability to show the interests of the user and the activities that were required to earn the badge leads to the "group identification" because the users can see who else has shared that experience. However they state that work should be done to examine if badging systems are engaging and motivating for all, because evidence suggests that they are not universally appreciated or understood.

A leaderboard is one of the game elements that can be applied to turn a non-game context into a gamified context. Butler[16] conducted a study on the effect of leaderboard ranking on the perception of fun. There were three conditions tested, one in which the player is at the top, another at the bottom of the ranking, and the last one were there are no other players in the leaderboard. The players were presented with a randomly selected leaderboard after being recruited using an online discussion forum. It concluded that being ranked at top of the leaderboard polarized the perception of fun, with some players having fun and others not so much. To the contrary, being ranked at the bottom, centralized the perceptions of fun. Also, players in the losing version had a replay percentage more than the double of the blank version, even though with the responses given by the players in a post-game survey, the players in the losing version only responded that they would play again slightly more than the two other

(a) Friends Leaderboard.

(b) Top Leaderboard

Figure 2.3: The Rocket League leaderboards.

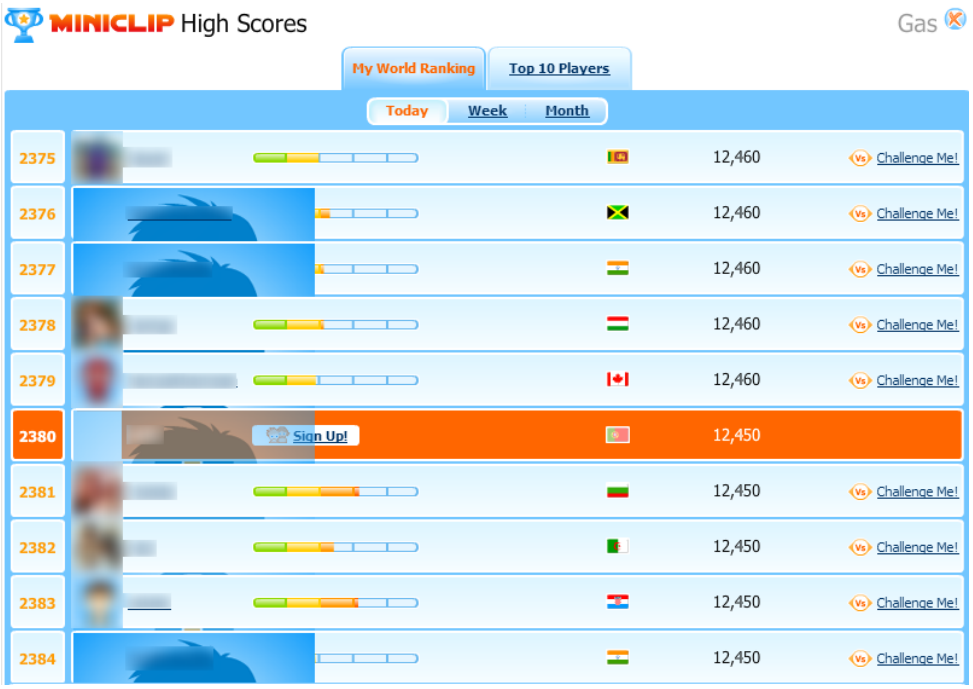


Figure 2.4: A Miniclip game leaderboard.

versions. One other result is that designing the game to be accessible to players also reduces the fun, while score does not seem to have an effect on fun rating. Finally, they point out some potential biases like the significance of the number of testers and they way they posed the questions.

Mekler *et al.*[2] used a image tagging platform to study if points, levels and leaderboards do any harm to intrinsic motivation. They measured the performance by tracking the number of tags generated by the participants and the time spent on the annotation task. Their hypothesis is that in a gamified context, with the elements they want to test, will significantly boost the performance of the annotation task, and decrease the need satisfaction and intrinsic motivation of the participant. The results show that indeed there is an increase in performance of participants in a gamified condition, because they generate more tags, than in the control condition. Participants in the leaderboard and levels condition significantly

outperformed the ones in points condition - see figure 2.5. They discovered that their performance decreases over time, but “participants’ performance in the leaderboard and levels conditions declined more slowly than in the other two conditions”. The hypothesis that it increases motivation could not be confirmed, since they not found any significant effects on the participants’ intrinsic motivation. The mere addition of these game elements to a non-game context is not enough to facilitate intrinsic motivation.

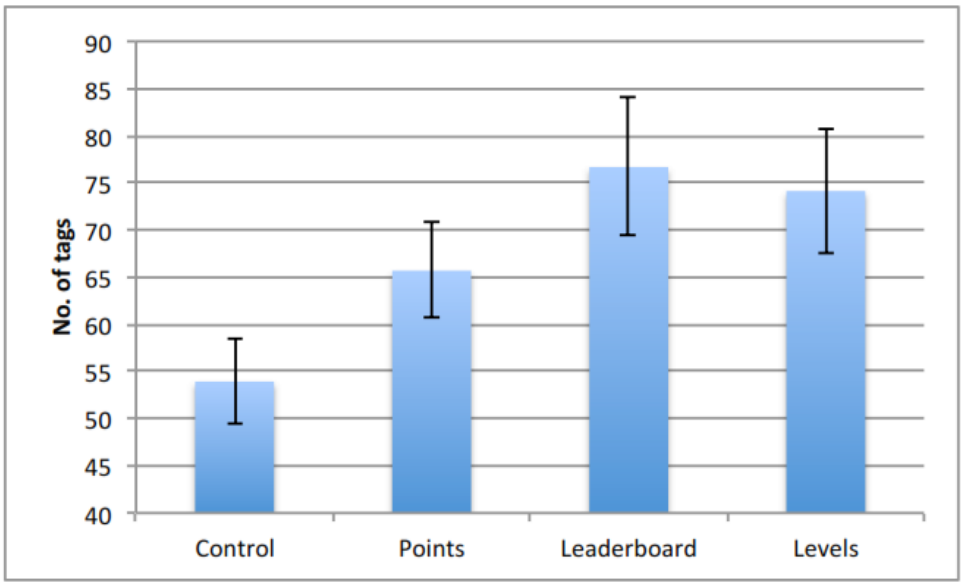


Figure 2.5: Average number of user-generated tags per condition. Error bars indicate 95% confidence intervals. (taken from Mekler *et al.*[2])

Many games use badges as one of the awards to the players and it is now used on online services like Stack Overflow. Denny[17] conducted a study with more than 1000 students with the intent of providing “empirical evidence of the positive effects that badge-based achievements have on user participation”. The experiment used PeerWise, an online repository which allows the students to author and answer questions, with an implementation of a badge system that was previously developed for the system. Students were alerted when they received a new badge. They were divided in two groups, one with the badge system enabled and the other with no badges. In the four weeks of the experiment the students authored 2620 questions and submitted 95685 answers. The results show no significant difference in the number of questions authored by students in the two groups, but the students in the badge enabled context answered on average around 20 more questions, with the same correctness rate of 69%. He recognizes that the data of the study can be skewed because the “most active 10% of students [of both groups] submit approximately one third of all of the answers”. In response to a post experiment survey the students indicated that being able to earn badges was enjoyable, with more than 60% of responses agreeing to this and no difference between genders.

Costa *et al.* [18] used leaderboards to tackle the issue of being late to meetings and try to improve motivation. Their study was conducted during two weeks at Games and Media Entertainment Research Lab at University of Ontario. They measured the arrival times on daily meetings and used those times to calculate the score on leaderboard. From their observations they argue that using a leaderboard influenced the interest of the participants, having the participants who were already interested becoming more interested and those who were not becoming less interested. It was a hindrance because some participants had to stop what they were doing and feel obliged to go to the meeting to get a good score, which made the meeting feel like a problem. The leaderboard influenced the behavior of the participants who started to compare punctuality. Even though they started to compare the punctuality they feel like

their own punctuality did not improve, but made the punctuality of others improve. They identify some limitations of the study, like having no control group and having only collected data for two weeks.

Social comparison can change the way we act. Christy and Fox[19] used a leaderboard in virtual class room to test effect of comparison of with other people in woman's math performance. They exposed women to a female-dominated and a male-nominated leaderboard and after taking the math quiz, women placed in the female-nominated leaderboard performed worse but demonstrated stronger academic identification than those in the male-dominated leaderboard. It is suggested that woman in female-dominated leaderboards made compared themselves with other women in a higher position of the leaderboard, but in the male-dominated leaderboard did not make any comparison or just with females that were in a lower position of the leaderboard. Since the study was conducted on women its not possible to generalize to men.

Following their study on the image tagging platform, Mekler *et al.*[20] performed other study to really understand the effects of each game element in the performance and intrinsic motivation. This study used the same image tagging platform, but now splits the participants by causality orientation, having autonomy oriented individuals vs control oriented individuals. The applied measure is the same for performance, the Intrinsic Motivation Inventory is used to assess the intrinsic motivation and satisfaction of the need for competence, and finally to distinguish the causality orientation they used a questionnaire, the 12-vignette General Causality Orientations Scale[14]. They found that the participants' causality orientation did not influence the effects of the game elements and the results are similar to the previous study, however a significant effect on causality orientation on tag quantity was found, because the Autonomy oriented participants generated on average more tags than the control oriented participants. The effects on intrinsic motivation were also similar to the previous study, but the results show that the autonomy oriented participants were more intrinsically motivated, as expected.

Sun *et al.*[3] studied the effect of leaderboards on the satisfaction of the players. It was unclear if a counterfactual thinking played a role for leaderboard positions, comparing to some studies on the Olympics. The participants played a web browser game and responded to a survey to determine how satisfied they were about their scores. It was revealed that players ranking second, fourth or seventh in the leaderboard were more satisfied than the rest (figure 2.6), "suggesting that counterfactual thinking may work differently in digital games than event-based competitions like the Olympics". They noticed that there may be other mechanisms at play in addition to counterfactual thinking, because the experiment revealed that "staying out of the bottom three was as important as being in the top three" and "counterfactual thinking alone would suggest that 8th place players would be more satisfied than 7th place since they were on the cusp of placing out of the bottom three". Even through current leaderboards allow the players to view their performance in different ways it was suggested that games should present leaderboards in which the player is in second, fourth, or seventh place, creating the enjoyment and motivation to play. This creates a problem, because you cannot constantly display the player at the same position every time, because it could be demotivating.

2.5 Discussion

Overall the studies agree that gamification has benefits but also some problems. In the article by Deterding[5], it is suggested that some users fail to engage with gamification and the gamified context should already present intrinsic motivation for the users to engage with it. Knaving and Björk[6] reinforce that idea by saying that the gamification should not be presented as an extrinsic motivation, in the case it has little or no reward in real life, which could harm the intrinsic motivation. One other

[14] http://www.selfdeterminationtheory.org/general-causality-orientations-scale/

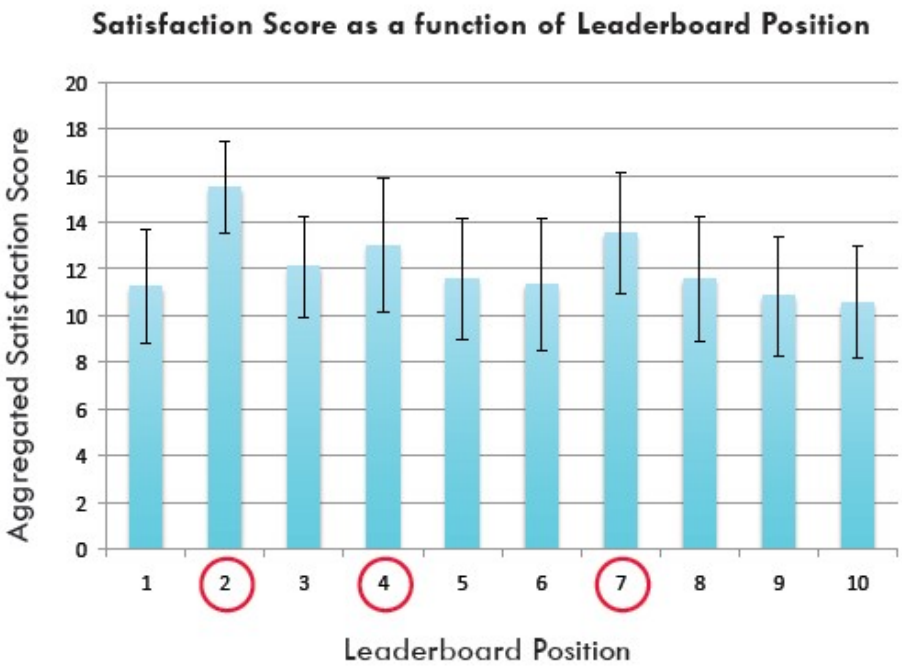


Figure 2.6: Graph of satisfaction score as a function of leaderboard position. (taken from Sun *et al.*[3])

problem presented by Hamari *et al.*[7] is the increased competition when in the presence of some game elements. This problem could be solved by applying the group segmentation suggested by Meder and Jain[8], where players are attributed different kinds of game elements in the gamification context. In the context of PCM this can be solved with the implementation of the way adjust information displayed to each student.

From the analysis of the studies we can try to consolidate some of the results and ideas to improve the gamification elements of PCM. The leaderboard is the main element used in displaying how the student is doing and identifying the ways it can be presented is important to improve students' motivation. Table 2.1 shows a summary of the effects and findings of the studies.

Table 2.1: Leaderboard Effects/Findings

Type of Leaderboard	**Placement**	**Constraints**	**Effects/Findings**
Unknown	Top		Polarized perception of fun
	Bottom		Centralized perception of fun
Top 10 players	Any position	Women Male-dominated leaderboard	Good math grades
	Any position	Women Female-dominated leaderboard	Bad math grades Good academic identification
	2nd, 4th or 7th		More satisfaction

Sun *et al.*[3] suggested the presentation of leaderboards where the players are in second, fourth or seventh position, however this cannot be applied in the context of PCM if a student is in first position of all students, because we cannot create a fictitious student, however it is possible to be implemented for the other students, by displaying them in a fake position. One interesting thing to explore is the polarization effect of being on top of the leaderboard, found by Butler[16] because if we can isolate the people that like to be ranked on top we can show this kind of leaderboard to them, while displaying other types to other players. Following the findings of Christy and Fox[19], when the leaderboard is shown to women it is difficult to figure out if they should be presented with a male-dominated leaderboard because even

though they had better grades in the math test the women in the female-dominated leaderboard had better academic identification.

Badges seem to be one of the most common elements implemented in gamification. In the system studied by Montola *et al.* [14] the users where mostly indifferent to the achievement system but it helped them learn the system. However the study by Antin and Churchill[15] argues that badges are a good way to motivate people, define clear goals to be achieved and create group identification with improves cooperation. In the context of PCM, badges have these functions, they define the tasks that have value to the course. One thing that is mentioned in the studies, and could be added to PCM, is the notification of users when they get a badge, which can cause instant gratification and allow the user to focus on other task to perform and earn the badges.

Taking into account these studies, one of the requirements for our system is for it to be flexible enough to accommodate all these types of leaderboards and game elements. To take it a step further, it should be able to be easy to configure the system with the game elements we want and at the same time have control of what we want to have displayed in relation to them.

Chapter 3

SmartBoards

SmartBoards is the new web solution that replaced the old static pages used for the gamification of PCM. It is a dynamic web solution that is highly configurable and allows the administrators of the system (the professors) to adjust the different aspects of the presentation of student information, like the way the leaderboard is displayed, throughout the semester. It is also easier to develop new gamification elements, and provides a new look and better interaction for the users. This easy modifiability and usability is what distinguishes it from the old system.

3.1 Old System

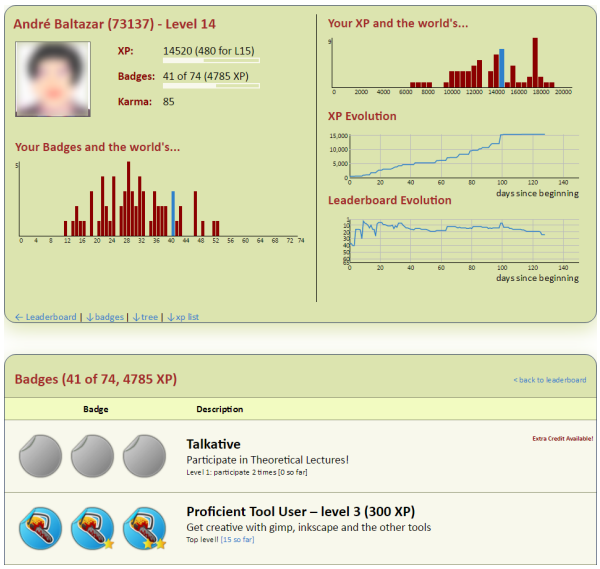
The old system build using a single Python script which generated static pages for the professors and students. It has to be manually run everyday to generate the pages. The script fetches information from text files that define the various elements of the gamification, a log from Moodle, and a Google Spreadsheet used by the professors to specify the attendance to classes, laboratory grades and other aspects of the course that need to be manually assessed.

The entry point for the system is the leaderboard page that displays the students ordered by Experience Points (XP) (which determines the final grade). This leaderboard contains columns for position, photo, campus, name, experience, level and number of achievements/badges of the students. However, it lacks some features that could benefit the users, like an ordering mechanism, or different views to see the information in a different way. This is what the SmartBoards system brings.

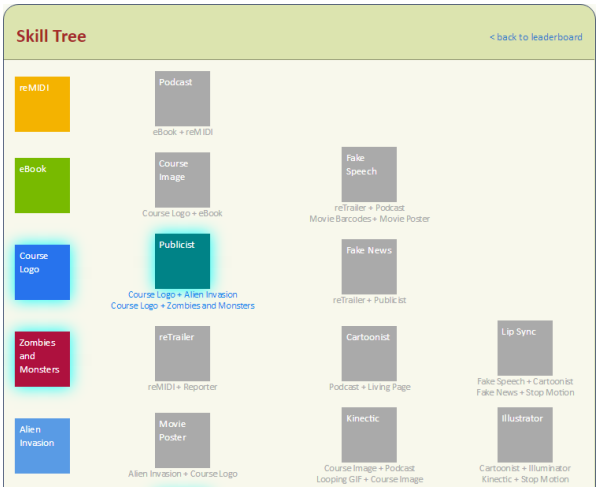
Upon clicking on one of the students in the leaderboard (shown before in figure 2.2), we are redirected to the personal profile (figure 3.1a) of the student, which contains the information about the student. It shows a few charts that contain the evolution throughout the semester and distribution of badges and XP. It has a table of all badges that can be achieved with their respective descriptions and progress, a skill tree (figure 3.1b) with the different skills that can be unlocked by doing some work, and finally the XP List (figure 3.1c) that shows when and how many XP the student was rewarded for something she/he did. All this information is relevant to the student, but it may be necessary to display different information to different students, for example, the student could be alerted that the time is running out to submit an assignment, which would not be possible to do with static pages, where there is no authentication mechanism to keep track of the users.

The system also has some pages that display the description of skills, with examples of possible submission, a page per badge that shows who has the badge and the date they got it for the different levels within the badge and a page that displays every award of XP given to students.

All these pages are static and hardcoded into the script, which makes them difficult to change. The



(a) Student Information and Badge List.



(b) Skill Tree.

(c) Awards List.

Figure 3.1: Different parts of a Personal Profile

SmartBoards system, makes it easier for the professors to change the way the pages are displayed and the information that is presented. It also allows professors from other courses (other than PCM), to use the system, because of the modules that allow to extend the system easily for whatever it is necessary to display.

3.2 Overview of SmartBoards

The SmartBoards system brings to the course a refreshed interface and new features for the students and professors. It uses modules to add new game elements, new ways to display information and other functionalities It also has the capability of changing the theme of the system, to change the color scheme, images and form of presentation of the different elements in the pages.

It's main focus is in the presentation and not in the computation/operation of the gamified experience. To achieve this we rely on the old script to fetch and transform the information, even though the new system knows almost everything about the "configuration" of the gamified experience in order to display

the information to the users.

Even though the focus is on the presentation, the system is more than a replacement for the static pages of the gamification of PCM. It can be used for any other purpose, as its parts can be easily replaced, for example, the login system could be easily changed to authenticate users using another method.

The login system used to identify the users requesting the pages allows us to fulfill the goal of the project, which is to allow differentiated forms of presentation to different students/users. The authentication is done using a Fenix Application API[1] which provides the information needed for the identification of the user. The Centralized Authentication System (CAS)[2] was also considered as a login method for the system but would require someone at IST to approve the application, while the Fenix API is available for everyone, that wants to develop applications with interaction with the Fenix system, in this case for authentication of the user.

Each authenticated user has roles that are assigned to them and specify which permissions they have in the system and the ways the information should be displayed. Many aspects of the system can be configured by the professors using a Settings back-end, which is one of the new features.

3.2.1 Settings

As previously said, the Settings back-end enables the configuration of all aspects of the system, from skins/theme customization to assigning roles to the different users. This is important because it gives the professor control over the course to manage the different aspects of the system in "real-time" without having to run a script and regenerate the pages to reflect that change. The Settings are actually split in two different pages, the General Settings, for the configuration of the system, and the Course Settings with allow for the configuration of course specific features. The general settings can only be accessed by administrators of the SmartBoards system and allows for the management of Courses (create, enable/disable and delete). The Course Settings is where the professors are able to create and assign different views to each student or a group of students. A view defines the way pages are presented across the system and can used to affect the engagement of the student. The Leaderboard and Profile pages, are just two examples of these views, because the system allows to create pages with anything we want due to its modules allowing to define any code in PHP, HTML, JS, or CSS for the system. Many other features are available in the Course Settings pages like the configuration of landing pages (where the user gets redirected when accessing the course), module management, create/delete quests and generate the external API Key.

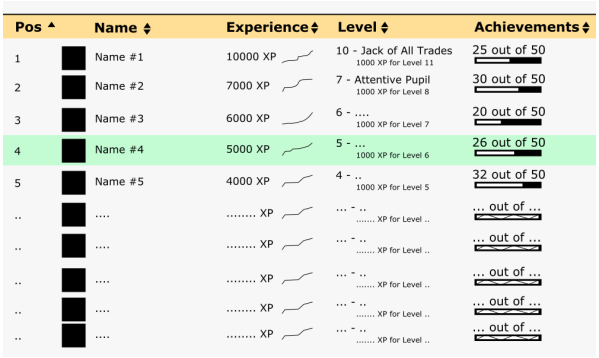
3.3 Prototypes

In order to figure out how the system would work, we created two prototypes that showcase the main pages of the SmartBoards system.

3.3.1 Non-Functional Prototype

A non-functional prototype (figures 3.2 and 3.3) was created using an image editing tool to provide an initial look of system and a starting point for how the system should be organized. These prototypes bring to the SmartBoards the leaderboard and personal pages, by redesigning them in a way that allows for the customization and extensibility of the system.

[1] http://fenixedu.org/dev/tutorials/use-fenixedu-api-in-your-application/
[2] http://dsi.tecnico.ulisboa.pt/servicos/autenticacao-e-acesso/autenticacao-cas/

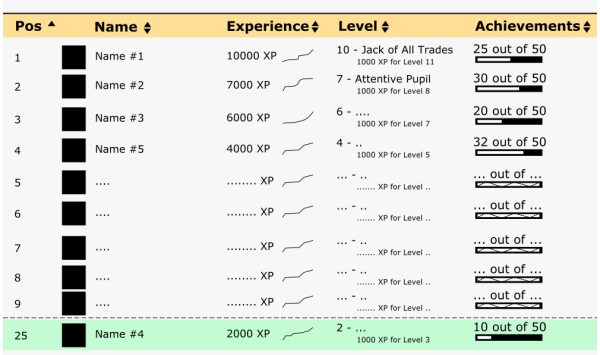


Pos	Name	Experience	Level	Achievements
1	Name #1	10000 XP	10 - Jack of All Trades 1000 XP for Level 11	25 out of 50
2	Name #2	7000 XP	7 - Attentive Pupil 1000 XP for Level 8	30 out of 50
3	Name #3	6000 XP	6 - .... 1000 XP for Level 7	20 out of 50
4	Name #4	5000 XP	5 - ... 1000 XP for Level 6	26 out of 50
5	Name #5	4000 XP	4 - .. 1000 XP for Level 5	32 out of 50
..	....	..... XP	... - .. .. XP for Level ..	... out of ...
..	....	..... XP	... - .. .. XP for Level ..	... out of ...
..	....	..... XP	... - .. .. XP for Level ..	... out of ...
..	....	..... XP	... - .. .. XP for Level ..	... out of ...
..	....	..... XP	... - .. .. XP for Level ..	... out of ...

(a) Course Leaderboard with XP evolution sparkline - default view.

Pos	Name	Level/XP	Achievements	Latest Award
31	Name #1	10 - 10000	Proficient Tool User - L3 4 people have this badge	Quiz 4 - 750 XP
32	Name #2	7 - 7000	Attentive Student - L2 Yay! This is a unique badge!	Quiz 4 - 750 XP
33	Name #3	6 - 6000	Lab Master - L2 3 people have this badge	- 100 XP
34	Name #4	5 - 5000	Proficient Tool User - L3 4 people have this badge	- 300 XP
35	Name #5	4 - 4000	Right on Time - L1 20 people have this badge	Quiz 4 - 750 XP
36	Name #11	3 - 3800	Right on Time - L1 20 people have this badge	Quiz 4 - 750 XP
37	Name #12	3 - 3500	Right on Time - L1 20 people have this badge	Quiz 4 - 750 XP
38	Name #13	3 - 3400	Lab Master - L2 3 people have this badge	Quiz 4 - 750 XP
39	Name #14	3 - 3300	Lab Master - L2 3 people have this badge	Quiz 4 - 750 XP
40	Name #15	3 - 3300	Right on Time - L1 20 people have this badge	Quiz 4 - 750 XP

(b) Course Leaderboard with unique achievements and latest awards - neighbors view.



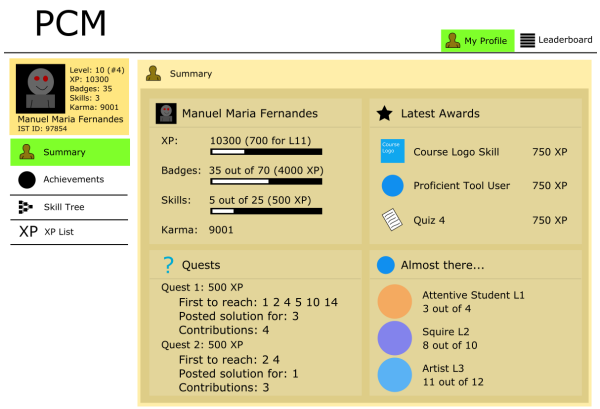
Pos	Name	Experience	Level	Achievements
1	Name #1	10000 XP	10 - Jack of All Trades 1000 XP for Level 11	25 out of 50
2	Name #2	7000 XP	7 - Attentive Pupil 1000 XP for Level 8	30 out of 50
3	Name #3	6000 XP	6 - .... 1000 XP for Level 7	20 out of 50
4	Name #5	4000 XP	4 - .. 1000 XP for Level 5	32 out of 50
5	....	..... XP	... - .. .. XP for Level ..	... out of ...
6	....	..... XP	... - .. .. XP for Level ..	... out of ...
7	....	..... XP	... - .. .. XP for Level ..	... out of ...
8	....	..... XP	... - .. .. XP for Level ..	... out of ...
9	....	..... XP	... - .. .. XP for Level ..	... out of ...
25	Name #4	2000 XP	2 - ... 1000 XP for Level 3	10 out of 50

(c) Course Leaderboard - top view.

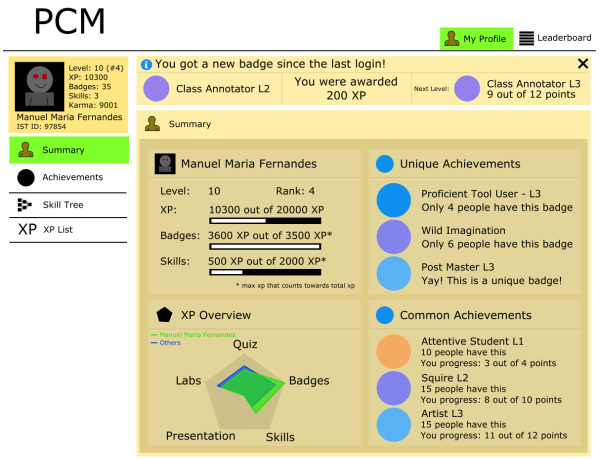
Figure 3.2: Leaderboard views of the non-functional prototype.

For the leaderboard page we created three figures (3.2a, 3.2b and 3.2c) that show some features of the SmartBoards, that were not present in the Leaderboard page of the old system of PCM. In the first of these two figures (3.2a) we can see a spark line representing the evolution of XP during the semester. This illustrates the ability of displaying charts in the Leaderboard. In the second figure (3.2b) we can see two new columns, the unique achievements, a column that contains the most unique achievements unlocked by the students and the latest awards column that show what was the last activity the student performed to earn some XP. This figure also illustrates one important feature in the presentation of the Leaderboard, showing what we call the neighbor view, that is, displaying only students that are in a certain range of position from the student viewing the leaderboard. In the last figure (3.2c) we have other view showing only the top students and the student viewing the leaderboard.

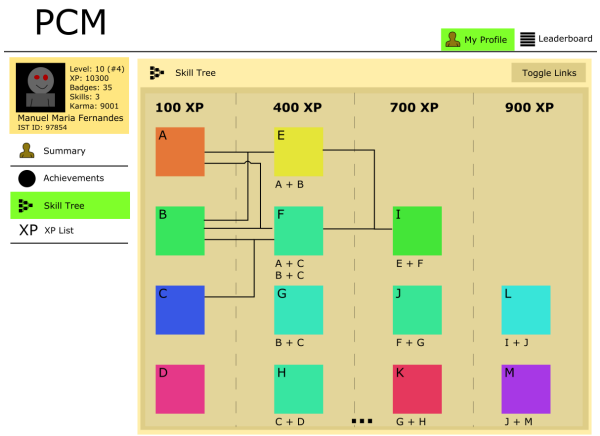
To have an idea on how the Profile page would look like we created four figures (3.3a, 3.3b, 3.3c and 3.3d), that not only show the new base components of the page, but their organization. In figure 3.3a we can see what we call a page block, which is sometimes a container of smaller blocks, with the summary of the student's progress and achievements in the course. We can see the character sheet block with some information regarding the student (XP, Badges, Skills and Karma). We can also see examples of blocks for the latest awards, that show a few of the latest things the student has done, the Quest block, a block that contains information regarding the student's participation in the quest, and a block that shows what the player is about to achieve. The figure 3.3b shows a block for unique achievements, a block with a radar chart with the different components of evaluation of the course and finally common achievements that other students have and the current progress of the student towards those achievements. Another page blocks can be added to the profile page, in the case of figure 3.3c we can see a page block with the skill tree inside. Finally in figure 3.3d we can see another page block, in this case for achievements.



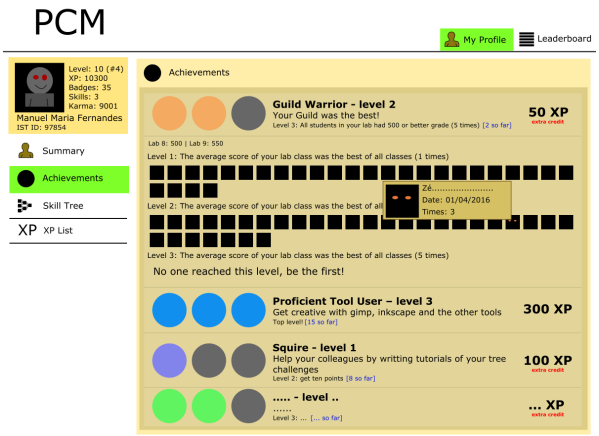
(a) A student's profile with basic information, quest information, latest awards and near completed badges.



(b) A student's profile with basic information and notification.



(c) Skill Tree in a student's profile.



(d) Skill Tree in a student's profile.

Figure 3.3: Profile Page of the non-functional prototype.

It shows the achievements, which ones are unlocked and the XP earned by completing them. It also shows a possible interaction by clicking on an achievement, the panel expands vertically showing how has got the achievement and when.

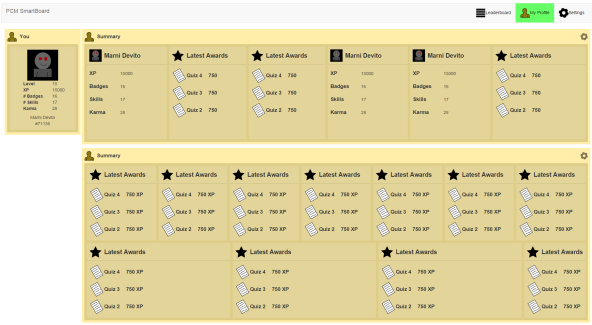
These views are what this non-functional prototype achieved and was the base for the functional prototype described in the next section.

3.3.2 Functional Prototype

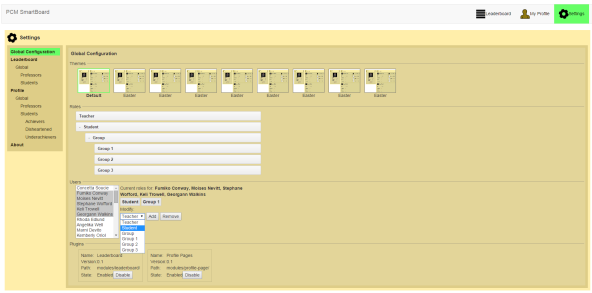
Taking into account all the views created for the non-functional prototype, a functional prototype (figure 3.4) was created using randomly generated data to show the possible interactions in the system and the tools the professors have to configure the course. Some basic blocks for the profile page were created for this prototype, to illustrate the configuration of this kind of page, and a way was also developed to edit the different views of the pages, that was not that good and the way to edit the leaderboard columns was too technical to use, requiring to change PHP code. This prototype already had a back-end to serve the pages and the concept of modules, however it disregarded completely how the different data of the modules/game elements were stored.

The leaderboard page (figure 3.4a) shows a page block containing the leaderboard with sortable

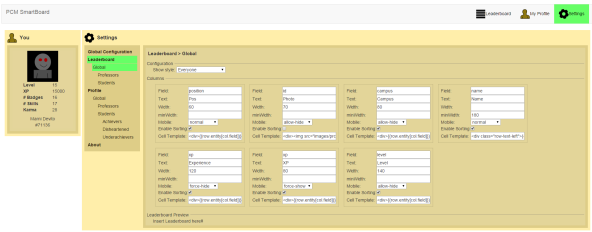
(a) Leaderboard page.



(b) Profile page with two page blocks.



(c) Settings page.



(d) Page to configure leaderboard view.

Figure 3.4: Functional prototype pages.

columns working. One feature implemented in this prototype was also responsiveness, allowing the page to be viewed in a mobile device correctly by removing columns to fit in the screen. In the profile page (figure 3.4b) we can see two types of blocks inside two page blocks. This page and blocks also adapt to mobile, shrinking and reordering to fit the smaller screens. Finally, the settings page showed who we would be able to configure the roles, themes, modules and views of the different pages of the system. These concepts are described in detail in the subsections of section 3.5. One problem found in this stage of development was the way roles are assigned to the views of the system, which lead to a simplification in the way they are done in the current implementation, and there were also some minor issues relating to the controls to assign and remove roles to each user. One other problem in this stage was in the page to configure the leaderboard (figure 3.4d) which was considered too technical to use by the users and should be replaced by something more visual, for example dragging new columns to the leaderboard.

This prototype used the technology described in the next sections, except for the table creation/sorting which used a AngularJS module called Angular UI Grid[3], which provides a easy way to create tables, the navigation bar used Twitter's Bootstrap[4] and it had no database. Taking into account the problems found in those pages and the disregard for the data storage and much of the back-end it lead into the creation of a new version from scratch, based on this version.

3.4 Architecture and Technology

The system can be logically divided in three main components (figure 3.5): front-end (the application), back-end (the server) and data source (the student information). These are the main components of the system, but it is also important to mention the modules. These are transversal to the system, allowing

[3]http://ui-grid.info/
[4]http://getbootstrap.com/

for the customization/extensibility of the system as described in section 3.5.6.

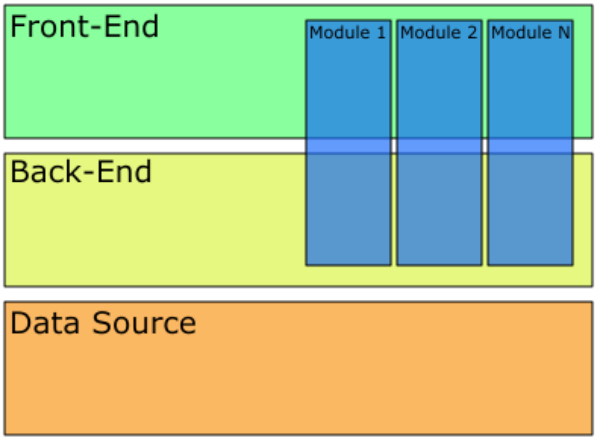


Figure 3.5: Main components of the SmartBoards system.

The front-end is what users see and interact with, which makes it an important part of the system, because having a interactive system with the ability of presenting different views to different users is the main objective for the development of the system. For its implementation we use the typical languages for web development, HTML, CSS and Javascript. To make it feel more like an web application, we use AngularJS[5], a JavaScript framework that provides some functionalities to aid in the creation of web applications. The front-end also makes use of JQuery[6] and some JQuery plugins, like tablesorter, and AngularJS modules, which make the implementation much easier. These plugins and modules provide functionality like sorting of tables and routing of pages within the application. In order to have a more structured code for the CSS, the implementation uses Less[7], a CSS preprocessor that extends the CSS language by adding variables, functions and other functionalities.

To serve all the pages and information to the front-end and process necessary requests for updates/changes in configuration, we have the back-end. It takes care of the logins for the system, which are processed using a Fenix Application, as described before, but can easily be configured to accept another login method. This part of the system uses PHP: Hypertext Preprocessor (PHP) because it was necessary to host the server for the PCM course, and PHP is one of the few languages that can run on IST's Sigma Cluster. This does not mean that it is difficult to run the system in any other server, because PHP is very widely used, and is supported to run in many different servers. For data storage in the back-end the system is using a few simple key-value databases, one for the general configuration of the SmartBoards system and one for each course the system has. With this, the system is able to support different courses without reinstallation, allowing for the simple creation of a new course with different game elements and configurations each semester. These databases were, at first, flat files managed by a simple PHP library called Flintstone[8]. The choice of this database library is due to the kind of information that the system needs to store, and the lack of necessity for advanced queries to be executed on the data. However this choice created response times and concurrency problems when accessing to the files. These are addressed in detail in section 3.5.2.

The data source of the system is where students' information comes from. In order to simplify the system it uses the data provided by the python scripts used by the old system, with some minor adjustments to output some raw information that was being used, but not outputted. The script uses

[5]https://angularjs.org/
[6]http://jquery.com/
[7]http://lesscss.org/
[8]http://www.xeweb.net/flintstone/

a variety of sources to get the information, from flat files, that contain the configuration of badges and other game elements, to some Google Spreadsheets, that contain the grades and awards given to the students. This information is then processed by the script and outputted to other files that are loaded into the SmartBoards system.

3.5 The System

The rapid development of the system was due to the intent of having a minimal working version ready for deployment for the course of PCM in the second semester of 2015/2016, which started in February 2016 and ended June 2016. The version deployed in the start had all the features of the old system, and took into account the problems detected in the creation of the functional prototype.

However, since the version was developed so fast in order to meet that deadline, this version had some problems that had to be tackled in order to proceed with the development. This was expected in the start and some minor problems were found but quickly corrected. An example of one such problem occurred because the development of the system was done using data from the end of the course of PCM from the previous year, and it caused some problems when it started fresh and progressed for the first days.

3.5.1 Database

In order to make the system flexible to new game elements, as discussed as one requirement coming from the related work, it was decided to use key-value databases. This type of database allows us to define arbitrary keys for each module that is loaded in the system, and don't be worried about altering tables, and all the other things required to make a relational database system work. To make this work, because of size and concurrency problems which are discussed in section 3.5.2, we split the information of the database between different files. These files form the database as a whole, and were created in a way that is easy to understand and use for the creation of the different pages of the system.

The courses available and the general information of the system is stored in configuration database which gives us an overview of which courses the system has, which ones are active, and some other aspects of the general configuration, for example, the theme (styling of the pages) of the system. To store the general information related with the users we have other file, which stores information like, name, email and username of the user. This allows to have a central file which has all the users that have access to the system, for authentication purposes, for example. Each course has its own file, which stores all the general information related with the configuration of the course, users that are enrolled, and the modules that are active for that specific course. To store the data related with each module and user, we have separate files for each of them.

This all comes together to form the database, which can be easily accessed by the modules of the system, as if the information was all in one file. The wrapper classes developed to access the database allow to access the database, change the information related to anything and save the changes automatically in the right file, without the developer having to bother which file it is. With this component that can access any field easily, we wanted the front-end of the system to be able to use this information when the implementation of the different views was created, so we created a data schema which provides all the fields available to be displayed in the front-end.

The data schema is a collection of all available fields that can be displayed in the system. This collection is stored in the form of a tree. Some of these fields are registered by the core of the system, which knows for sure what fields are available in a course with no modules enabled. When a module

is enabled, it registers its own fields (if any) to the data schema, which allows the user configuring the system to use them when creating the different pages. These fields are defined by the developer allowing a finer control of what can be displayed or not. The module could be storing more information, but for some reason, for example, security, it does not allow that field to be displayed.

The core of the system, defines the basic information for the user, in an map of **users**, indexed by the id and contains an object with fields for a student, which are common in the whole system, as the name, email, and Fenix username. To define the course related information, a object called **course** is also defined in the root of the data schema. This object only holds a map of **users** with the information of the users, that are enrolled in the course. The information in this object is the **roles** of the user in the course, the **campus**, **last activity** and an object called **data** which contains the user data for the different modules installed in the system. Each of the modules is allowed to define information as they please in this, data field of the user. Configuration information for each module, is stored in the root of the data schema, with the key **moduleData**, and is defined on a module by module basis, with the information that they need to store.

As said before, all this flexibility came at a cost. Some problems were found when the system started to increase the amount of data stored and many people were accessing the system concurrently. This concurrency created a problem on the database, which would make the whole system fail. Part of this problem was that the system needed to keep track of the last access of the student, which was saved to the database. This created a problem when the database, as single file, was written over and over again with a few megabytes of information. Each request to the system would cause a rewrite of the whole database and with other request underway the system would fail.

3.5.2 Solving the Database problem

The concurrency and system failure problems, referred before, which occurred due to the increase of the data size in the system, were approached by trying to migrate the database from the flat file system, to a relational database approach. This approach and other solutions are discussed in the following sections.

3.5.2.1 Relational database approach

Using the a relational database would make the data access faster, since database management systems are optimized to handle high amounts of data. To use this approach a fork of the project was created to try to use a relational approach.

In order to try to speed the transition, some research was done on relational database object mappers. These are frameworks that map objects into tables automatically which would make the transition faster because the system is using objects saved into files. If a one-on-one correspondence between the new mapped objects and the objects of the system could be achieved this would mean that only a few classes relating to the load of the flat files had to be changed.

Research on the object mappers was done. We see that all these mappers require the developer to run a command to create a stub file for the object mapper and when loading the module, it would require to run other command to create the proper database tables. This would make the creation of the modules much more difficult, since they are supposed to load on the fly, whenever the course administrator wants, even though they are not supposed to always be turning on and off, but it is possible to create something like that if needed. The modules should also be allowed to register new information on the fly, to make it easier to develop new modules. For example, if we are developing a module and want to register a new field, it should not require the developer to run commands to regenerate the stub files and update the

tables of the system. Simply defining that the new field exists, as is done with the flat file implementation, will make use of that field. The option of using a object mapper was ruled out, however it should have been given more thought. A possible new implementation using this approach is discussed in the Final Solution section 3.5.2.3 of this chapter.

With the option of using an existing object mapper ruled out, an attempt at creating a mapper which would allow the creation of fields on the fly was started. The fork of the project evolved to use this approach with some success. The mapper was able to load the objects and its field from the database, however the creation and update of the tables proved very difficult to do on the fly, leading this approach to a failure.

3.5.2.2 Possible Solution and Impact on Views

With the failure of the relational approach, a new solution was created using the flat files. A file manager was created that split all the objects into different files, in order to make them more accessible, and more easily updated, since each file had less data. The file manager worked as expected, allowing to read and update the information in any path, however this approach generated a lot of files, which made an huge impact on the loading times of pages of the system that required many different fields what were scattered across the files, which take time to load from disk.

To solve this, a selective split of the objects into different files was created. The system manages this split, creating files only for the main objects of the system which greatly reduces the load times of the views, however not enough to be instant, but taking only a few seconds to load a page that uses a lot of information.

There is another problem that comes into play here. IST's Sigma is slow accessing files, which makes the system go at least 2 times slower than other computer (at least the one used to develop the system). The files can now be updated faster, avoiding the concurrency, but the slow load times, makes for a deteriorated user experience.

To solve this, a final solution should be created.

3.5.2.3 Final Solution

Taking into account all the problems found and discussed previously a new solution should be created where it is easy for the developer to create new modules, easy to install the modules, has a great response time and is able to update the information efficiently.

Part of the problem is related to having the restriction of running the system on Sigma. If the system was under our control and we could install other software, a greater number of approaches would be available to choose from. For example, using other types of databases, like NoSQL[9] with a key-value interface, having an cache mechanism that lives across multiple PHP requests, or using other programming language and database approach all together.

Since we have this restriction a new solution should be developed with it in mind. Having a cache single file, compiled with the information that is not always updating is one possible solution to support the flat file system. This would maintain the system as it is, requiring some effort to have the big cache file update as few times as possible, while having a great hit/miss rate.

Another approach to the problem, respecting the restriction of using Sigma, is to go back to the relational approach using an available object mapper. This would require a trade-off between the developer running commands to compile the data types and simple programming them. The problem of installing the plugins and having them run the setup on the fly could be solved by keeping track of the plugins

[9] https://en.wikipedia.org/wiki/NoSQL

that had already setup the database to work. When the plugin had a new version, the setup should run again to make sure the data types are not updated. Using this system would make the developer's job a harder because it would also require to have some mechanism to update the database schema easily when developing, without having to be always changing the version of the plugin to make the system run the setup.

These approaches described above are the real solution for the problem. However with the lack of time to really change the system to implement one of these solutions, another way was found to increase the performance slightly. Since the problem of the sigma cluster is mainly due to the number of files accessed, the files were moved to a relational database as a BLOB in a table, indexed by the file name. This is a strange solution, but speeds up the loading of the page by a few seconds, however it is not a real solution and should not be used for the long term of the system. While this improves the speed, it is still a little slow to load the pages, compared to what would be expected. The testing of these system outside the sigma cluster was done on a high-end computer, with 4 GHz i7 with 16 GB RAM which is twice as fast at serving a page.

3.5.3 Settings

One feature of the system is the support for multiple courses. This allows the possibility of creating an archive of all the past years of the course, for example. The initial plan was to have one active at a time, but allowing multiple active courses is better than having to install a separate instance of the system to run other course in parallel.

The general settings (figure 3.6) of the system are only accessible by the administrators of the system and allows for an easy creation of a new courses. When a new semester starts, the administrator can create the course, based on the previous year (or a completely empty course, with no configuration) and load the students and teachers into it. This part of the system also allows to configure the theme of the system, which is shared between all courses. It has a page where the administrator can manage the users which allows to add and remove administrators and to invite users to access the system. There is also a page where the user can get information related to the External API.

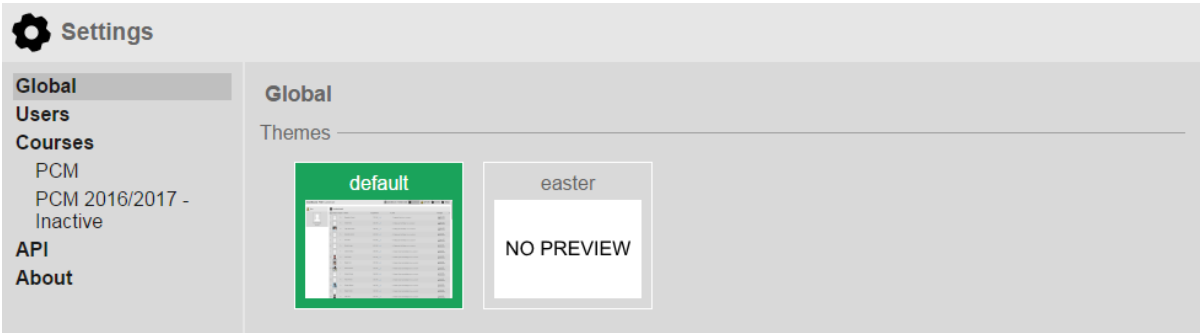


Figure 3.6: General Settings

The course settings (figure 3.7) allows to configure everything related to the course. One of the configurations is the landing page for the system, which is configurable per role, meaning that depending on the role that the user is inserted into they will go to a different page, when they navigate to the main page of the system. This settings page, also allows to configure the users' roles, and the hierarchy of the roles. The modules, which can also be managed in these settings, can register different pages to manage module specific configurations. Finally, the is also a page where the user can get information related to the External API, specify for the course.

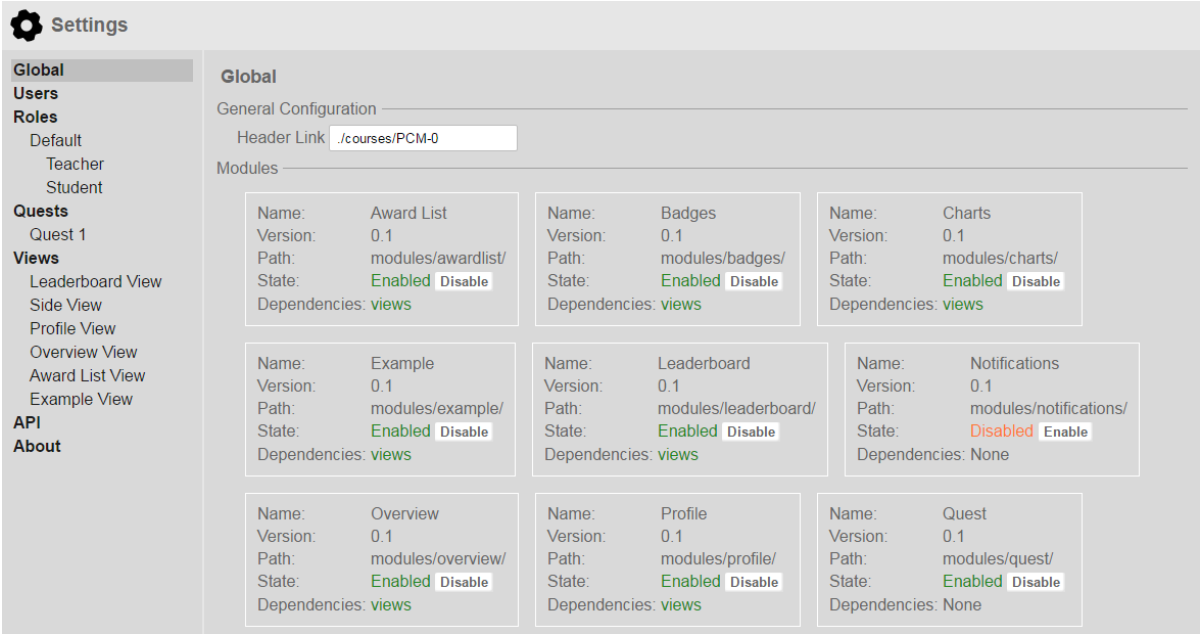


Figure 3.7: Course Settings

3.5.4 Themes

The themes allow for an easy configuration of how the course's pages should look like. They can define how the different aspects of the solution work in different devices, for example, the navigation bar in a desktop screen has the buttons side by side in the bar, while in a mobile screen the buttons will them show below each other. These can be used by the administrator to change the look of the platform, for example during the Easter it could be display Easter related images that could affect how the users interact with the system during the Easter break.

For now, there is only one theme and it is only configurable by editing a Less file. To create a new theme, all that is required is to create a new folder in the 'themes' folder of the system, with the name of the theme, with a 'main.css' file, which contains all the CSS rules for the theme, and a 'preview.png' file, which is a sample image of what the theme will look like when using that theme, or other way of identifying the theme being selected. In the future, extra functionality could be implemented to allow for an in-app configuration of the themes for a more user friendly way of editing them.

3.5.5 Roles

In order to differentiate the users in system we need roles. These roles are assigned by to each user, and each user can have multiple roles. These roles could be 'Teacher', 'Student', 'Group', 'Group 1', 'Group 2', etc, for example. These roles are organized in an hierarchy that defined the most relevant role.

Let's say a student is assigned the roles, 'Student', 'Group', 'Group 1', and we have defined the hierarchy in a way that the 'Student' has a 'Group' that for instance has the different groups inside. The most relevant role for that user will be the most specific one, 'Group 1'. In the case that the Views module (described in section 3.5.7) is enable, the role 'Group 1' would be the first candidate to be used to select the view presented to the user.

3.5.6 Modules

The modules are the most important part of the system. They are responsible for the extensibility of the system, define the pages of the system, the game elements, define pages to configure them in the settings page, and can even run code that change pages from other modules installed in the system, as is the case of the side view module detailed in section 3.5.8.3. Since they have such a wide reach in the system, the only page that is part of the core system is the “Settings” page, which cannot be disabled. All other pages in the system are defined as part of a module.

One other thing that modules can do is to store information for each user and see the information that other modules stored as well. In this way they can display that information, for example, there is a module that configures the existence of the game element ‘Experience Points’, and the module that displays the block of the side bar in the system, can see the data from defined by the module of the game element and display it. The modules can also define new internal API functions to be called by the web application, which allow to read, modify, or delete something in the back-end of the system.

To get started with the creation of a module, it is necessary to create a folder with the module name, which will contain everything related with it. Next, the main file of the module is defined starting with the string ’module.’ followed by the module name with the PHP extension. In this file a PHP class that is the main class of the module is defined and registered to the module loader, that has the registry of all loaded modules and their dependencies. Most of the modules use just this single file to control the system, for example, if they need to create a page which is controlled by the views module, this one file is all that is necessary, along with the declaration that the module depends of the Views module.

There are two types of dependencies between modules. A ’soft’ dependency defines that the module requires the other module to have some advanced features, but its not necessary to have that module enabled for it to work. The ’hard’ dependency requires the other module to be enable for it to work. One example for the necessity of this separation is that a module which uses the views to display pages could implement a fallback to a custom page with some reduced functionality, if the views module was not enabled. This is also used for example in the quest module, which does not use the views to display its pages, however it has a soft dependency from the views module, because it will load will load a template to be used in the views, if the module is present.

Some complex modules also require more than one file to run, these additional classes can be placed inside the module folder and are auto loaded by the core system when used. Along with these classes, some modules also require assets like JavaScript and CSS files, which can also be placed in the folder, registered using the main class of the module to be loaded in the page when necessary. These JavaScript files usually define some behavior to be used by pages related to the module, or to interact with other modules, for example again in the case of the views, to define a new part type.

To define a game element right now in the system, the main thing that is necessary is to define the fields in the data schema, which are then populated by the data coming from the old python script. However, a module is capable of doing more than just defining the schema, and in the future could integrate all the logic necessary to process the information directly from the source, without help from the script. This would allow make the system independent from the script, has each module would manage everything related to each game element.

The modules are allowed to define functions that other modules can use to interact with them. These functions are what powers the capabilities of the views to register new views, or define templates, for example. This interactivity between them, allows to define these modules that provide functionality that other modules can use easily and without having to worry about, for example, every aspect of how the views are implemented. Of course, this helps the implementation of the more simple modules, but the more complex need to be able to process requests from the pages. The allowed request functions are

registered to the internal API, which are processed by the core of the system, and only delegated to the module if the incoming request matches the registered function. Everything that requests something to the back-end needs to use one of these functions get the information, perform an update, or whatever the request does.

All of this defines what is possible to do with development to define new game elements, views, etc, in the system. However, one of the most important parts of the system its the configuration. To allow this configuration of the modules, they can define pages in the settings page with all the configurations available, which gives the user some control over how they work. The documentation with code on how to create the modules is available in Appendix B and in the documentation provided with the system.

3.5.7 Views

The views is a module that is at the core of the objective of view differentiation for each user. It allows each module to define the views for the pages they provide, for example, the profile module defines a profile view, which can then be configured in the settings page. As described in roles section (3.5.5), the views are assigned according to the most relevant role that exists for that user. In the case that no view exists for the roles, the user is presented with a default view. Another aspect of views not mentioned yet is the way users view other people's information. The system will allow to configure the views for how other people will see the information. The views are selected depending of what is being displayed, and who is seeing the page. For a simple page, where the information displayed in the page does not belong to only one person, for example, the Leaderboard, the only factor that is used to show the page is how is seeing the page. This allows us to differentiate between a Professor and a Student viewing the page, for example to show some other metrics to the professors. However, for pages that belong to someone, like a profile page, the selection is made using both the viewer and the user for which the information belong to. This allows us to have more control over these pages. The Student and Professor can have different pages (the selection is made first by the user which the information belong to) and then the viewer can have some differences (based on the role of the viewer). This allows for example to display private information in the student profile, if the page belong to them. Taking the example described in the roles' section. If a student has the role of 'Group 1' and another student is a group member, it will also have the role of 'Group 1'. In this case we could present more information on the page of the first student for the group member to see.

3.5.7.1 Layout Editor and Components

It provides a layout editor, that allows the administrator to edit the layout of the views. This editor is what truly accomplishes the objective of the project. To do this we it provides a mechanism to register types of components that are displayed within a view. There are four default types provided by the module: **value**, **image**, **table** and **block**. A value is a simple text value, the image is an image to be displayed, the table is a container that can include any the other components in a table format, and the block is contains and header with an image and title, as well as a body, which can contain any other components. Figure 3.8 shows these components.

These components were chosen, because they are the essential pieces necessary to create a basic page, like the Leaderboard and Profile Pages. The value and images are the most basic parts, that define the content of any page. The block allows us to have multiple parts, within its body, and organize them into sections using the header. And finally the table is, of course, used to organize the content in a table format, which is needed for many views. This organization in a table format could be done using multiple blocks, but the table offers an easier way to accomplish it.

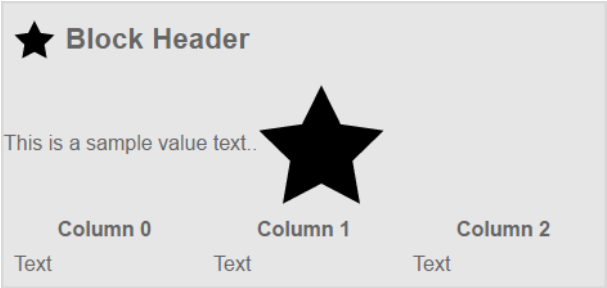


Figure 3.8: Block holding a Value, Image and Table

Each of these types of components have properties/fields that allow to configure the different aspects of how the part is presented, from styling and events (mouse clicks for example), to the actual content of the part. In order to configure these components in a more expressive way, it was required to create a expression language that is capable of manipulating text and numbers. It is similar to other languages (C and Java) when it comes to arithmetic expressions, and also allows to access the fields from the database and call functions defined by modules. An example of such expression is 'Hello {users[%viewer].name}!' which would be evaluated as 'Hello John!', for example, if the name of the viewer was John. This is possible by compiling the expression and then evaluating it. The evaluator automatically fetches information from the database to fulfill the expression, in this case the name of a user, with the IST Id that is stored in the variable 'viewer'.

To have a greater control of what is displayed, for example in a Text part where you want to get something from the database, we can use this specialized language. The language used was created out of the necessity to have a more powerful way to display text in the views, but it extended to used to define properties for other parts as well. It enables the administrators configuring the parts to have conditions, functions, data access and mathematics/logic/bitwise operators to display anything they want. The functions that can be called using the expressions are developed using PHP and registered using a module. They provide a way to extend the functionality of the expressions, for example, if we want to define a function to do some mathematics operation like the sine function, we could implement the function in PHP mapping it to the PHP build-in "sin" function and then call it in an expression. A description of the full capabilities of the Expressions, its design choices and problems, can be found in section 3.5.11.

3.5.7.2 Data Structure and Inheritance

To support all this in the editor, a data structure was created, to save this information. One obvious way to create this structure if by following the approach used by HTML and use a tree to save the elements of the page, which can then be translated directly into HTML elements. This is the approach used for both the editor and the render of the final page, even though the HTML elements created for the editor are quite different, displaying only placeholders for what the information will look like. This structure is composed of JSON objects organized in a tree, which can also be interpreted by PHP and then processed. Each of the object has the type of part it represents and its properties, as well has the children parts if they can contain any. The graphical representation of the data, used to represent a block with header and a text inside the block, can be see in figure 3.9.

Following the creation of the layout editor, an inheritance mechanism was implemented. This allows for a view to inherit parts from other view, which makes sense in the case of the differentiation of the views. The professor can configure a general view, of how everyone will see the view, and then from that view specialize for each case the differences for the specific case. In the case a view involves two users, for example the Profile pages, which have the viewer of the page, and the "owner" of the profile, the

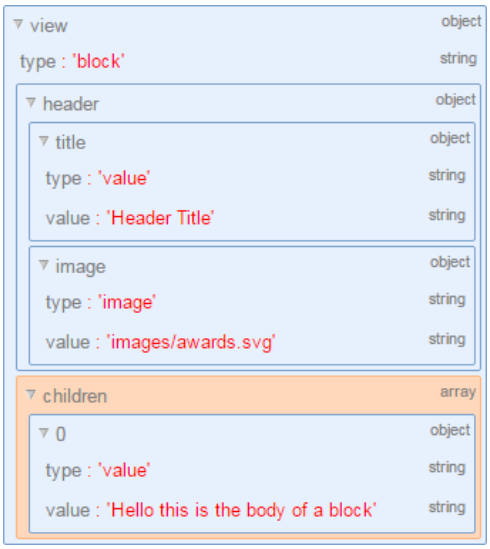


Figure 3.9: Data Structure

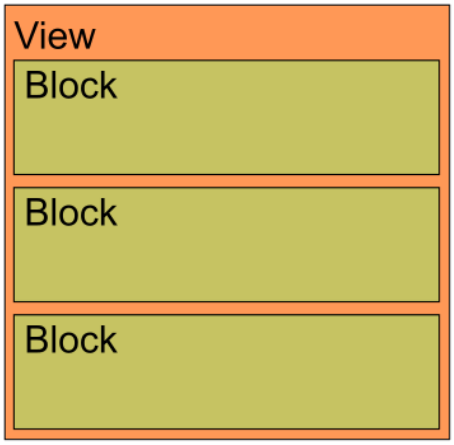
inheritance works for both users, first selecting the best match for the profile user, and then the viewer.

The data structure used to hold the information of the view was not appropriate to solve the inheritance. Each part needed to be uniquely identified, to be able to inherit some parts from others, and have everything update dynamically. A unique identifier was created for each part, based on a random string. The view tree is broken down into a part list with links between each part, so that it can be all be recreated using the same parts. When a new view comes along that extends the other view, it simply needs to point to the parts an reconstruct itself to be able to be processed and displayed. The sub-figures of figure 3.10 describe this process. Figure 3.10a shows a tree of the view, as used without inheritance. To allow the inheritance we introduce the unique identifier, represented by the 6 hexadecimal numbers in the figure 3.10b. The process of inheritance then starts with figure 3.10c, in which we see the view broken into pieces for the part list. These parts can then be used by a new view as seen in figure 3.10d where the a new view is introduced and has links to parts of other view. Finally to have a view ready to display the view gets reconstructed which results in the view tree represented in figure 3.10e, which can be displayed by the view renderer.

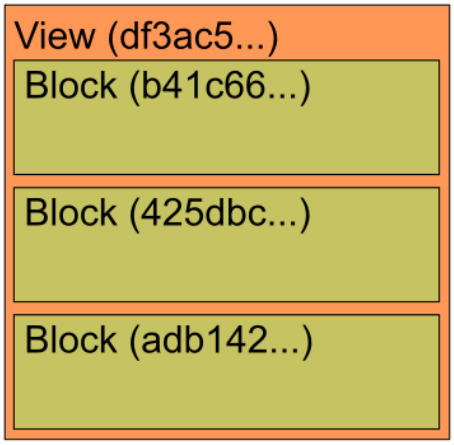
3.5.7.3 Properties of the Components

The properties available for the configuration are divided into three main groups, the **Part Specific**, which are defined by each part registered in the views, the **DOM Manipulations/Conditions**, which define manipulations of the part, like repeating it, and finally the **Styling/Misc** which contain other properties. All these properties are enabled/disabled on a part by part basis, which allow a greater control of what can be done with the different parts.

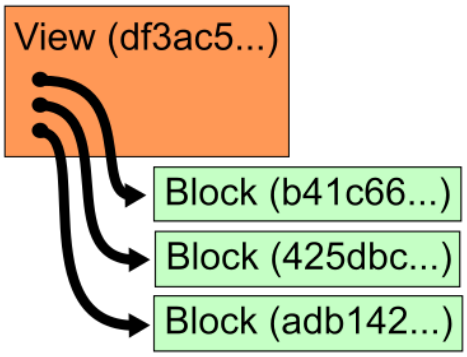
In the **DOM Manipulations/Conditions** group, there are two properties, the **If** condition, which allows to hide the part based on a condition, and the **Repeat**, which allows to repeat the part when necessary. The If condition, when enabled asks for a expression, that when evaluated as true, will allow the object to be displayed in the page. The Repeat property, repeats the part based on some container of elements, asks for an expression for those elements, and a key to define a variable which will be used to store the information of the element. This variable is accessible in the same part and its the children parts, if the part is a container of other parts. It is very useful to use on two sub-properties of the repeat, which are a **Filter**, which provided with an expression will filter out the elements, and a **Sort** property that allows to sort the blocks based on some value. Even though the filter seems similar to the If condition, they



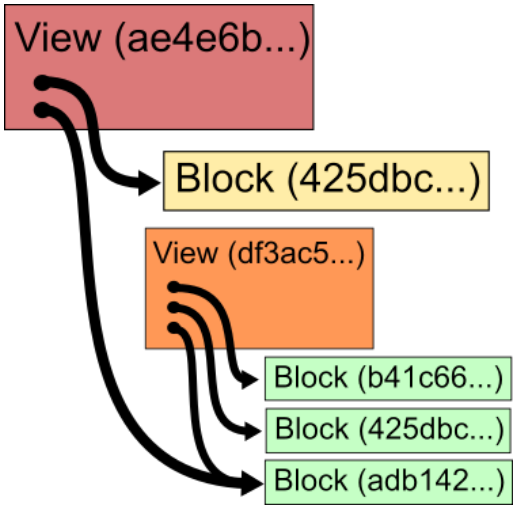
(a) View Tree



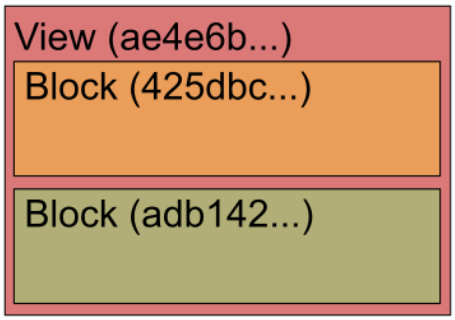
(b) View Tree with unique identifier



(c) View Tree broken into parts



(d) View Tree of a new view with inherited parts



(e) View Tree for the new View

Figure 3.10: View Inheritance process

do not operate at the same time. The filter is 'executed' before the sorting and the 'if' after it, which can increase control of the view, and even though the if seems to be useless in the case of a repeat, it can be necessary when the repeat is not enabled.

To enable styling and interactivity with the view, we have the **Styling/Misc** which define the properties necessary to make this work. The first property available is the **Style**, which allows the user to specify any CSS rules to be applied to the part. Next, we have the **Class** which similarly to the previous one, is related with styling and allows to specify any number of predefined CSS classes to be applied. The next two properties define interactivity with the part, they are the 'Angular Directive' and the 'Events' properties. The **Angular Directive** allows to specify an AngularJS directive to be used by the part, to provide some custom JavaScript functionality. The **Events** allow to define angular expressions that should be called for the different DOM events. This makes use of the Angular Directives which define the functions to be used here. Finally the last property is the **Data**, which allows to define variables based with the values of expressions. These variables have two types, 'JavaScript' or expression variables, the first one is used to get information that can be used by the JavaScript for example for the events, and the second one defined variables in the expression compiler that can be used by child parts in the view, just like the repeat does. This is particularly useful for caching of the values, or to define 'constants' which can be used in other parts.

As said before, the views module provides four default types of part to be used in its views. The 'value' and 'image', which represent text, and images, have two part specific properties, the value of the expression to be represented, and a property that allows the user to turn the text or image into a hyperlink. The 'block' part is a little more complex, it is used as a container for other blocks, and has one property that allows to enable/disable the header of the block, which is composed of an image and a title, which is a 'value' part. A sub-property is available to allow the block to be considered a page anchor (the user can jump directly to the block using an hyperlink), if the header is enabled.

The 'table' part type is much more complex than the other parts. It allows the user to configure the columns and rows of the table. It is possible to add header rows, which have a different predefined style. This table type allow the use of the properties available for configuration were necessary, and makes sense. It also defines a two part specific properties for itself, the first one which is Filter, allowing the user viewing the final page (not the administrator editing the page) to filter the rows according to some variable which should be defined in every row, in its Data property. The second property available here is a sorting feature which can also be used by the viewing user, allowing them to sort the columns of the table. This sorting capabilities can then be controlled on a column by column basis, using the Sort property available in each column settings, that allows to predefine the sorting of the column, or disable it.

3.5.8 Sample Views for Gamification

In the next sections we present some of the views that show the functionality of the views. These were implemented for the course of PCM.

3.5.8.1 Leaderboard

As talked in previous sections, the leaderboard allows the students to view their ranks in comparison with other students. One of the things that this project wants to improve in relation to the old system of PCM is the way that the leaderboard is presented, in order to achieve our objective of allowing differentiation. To do this we make use of the views described in the previous section, which allows for the differentiation of information presented to students.

In the prototype the leaderboard was implemented using the AngularJS's module Angular UI Grid which at the point of having implement a way to configure the leaderboard would make it a very hard task. It allowed to sort the table and hide the columns manually, however for the task of configuring the columns it would have made the system more complex since the UI Grid module does not use the HTML's table elements to create the table, and manipulating the table would be a hard task.

For implementation of the first version of the system we used unofficial fork[10] of a JQuery plugin called tablesorter[11]. It provided the required sorting capabilities, but creates the problem of having to construct the tables which the UI Grid module took care of. However, that was not a very difficult task, and the simplicity of the HTML code payed off, because we could use other plugins to manipulate tables as well. One of those plugins was the dragtable[12] JQuery plugin, which provides a way to drag and drop columns in a table. That is a plugin that makes configuring the table a little less technical and allows the users to see immediately the results of their configurations. However other problems arose in terms of performance and this had to be changed.

One way to create responsiveness when using the leaderboard in smaller screens is to hide columns and display information in a way that will fit in the screen. For example, in the case of charts, we can apply semantic zooming and get rid of axis and labels to make them fit he screen. The current implementation does not implement these features, but it was present in the prototype, which had part of these features, hiding columns if the screen was small, however it was a change made to the UI Grid that the current implementation does not use.

For the default template of the leaderboard (shown in figure 3.11), the sparklines discussed previously were implemented, to display the evolution of the players XP, all the other columns are very similar to the old system, but can be fully configured with the power of the views, which provides the capability of sorting for other things except XP. The views also allow to create different types of leaderboard, like encountered on during the research, the top 10 or the neighbor view (users with similar XP to the user viewer the leaderboard). To construct this, we use a table, with the Repeat property on its row. This allows us to have a row for each student in the course. We then use the variable with the student being repeated to get the information to be displayed in the different columns.

3.5.8.2 Profile Pages

The 'Profile Pages' module defines another view for the system. These profile pages display information for each user of the system and as previously talked these pages also needed to be configurable, so this module also makes use of the Views module.

For the default template of the profile page of a student (part of which shown in figure 3.12), we have four main blocks: summary information, achievements/badges, skill tree and award list. The summary information blocks are the ones that display information like the users XP, Level, number of achievements/badges earned, and information from other game elements. These also include charts that display this kind of information. The achievements blocks display information related to achievements earned, what is the necessary to accomplish to earn an achievement or who has the achievements. In the case of the skill tree and award list blocks these display information of the skills and awards earned respectively. This template is constructed using the different properties available, almost all of the properties are used to create the different blocks. For example, in the skill tree, is implemented with multiple Repeat properties, the first repeat is used to repeat for each tier available, the second displays the skills inside the tier, and than two others are used to display the dependencies. The figure 3.13 shows the different parts in the editor that are used to construct the view.

[10] https://mottie.github.io/tablesorter/
[11] http://tablesorter.com/
[12] https://akottr.github.io/dragtable/

Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
1		A	Manuel	20835 XP	20 - Science God 165 for L21 at 21000 XP	63 out of 67
2		A	Manuel	20535 XP	20 - Science God 465 for L21 at 21000 XP	50 out of 67
3		A	Manuel	20325 XP	20 - Science God 675 for L21 at 21000 XP	37 out of 67
4		A	Manuel	20285 XP	20 - Science God 715 for L21 at 21000 XP	49 out of 67
5		A	Manuel	20240 XP	20 - Science God 760 for L21 at 21000 XP	50 out of 67
6		A	Manuel	20230 XP	20 - Science God 770 for L21 at 21000 XP	53 out of 67
7		T	Manuel	20150 XP	20 - Science God 850 for L21 at 21000 XP	46 out of 67
8		A	Manuel	20090 XP	20 - Science God 910 for L21 at 21000 XP	39 out of 67
9		A	Manuel	20082 XP	20 - Science God 918 for L21 at 21000 XP	41 out of 67
10		A	Manuel	20050 XP	20 - Science God 950 for L21 at 21000 XP	42 out of 67
11		T	Manuel	19900 XP	19 - Professor 100 for L20 at 20000 XP	52 out of 67
12		A	Manuel	19868 XP	19 - Professor 132 for L20 at 20000 XP	40 out of 67

Figure 3.11: Leaderboard

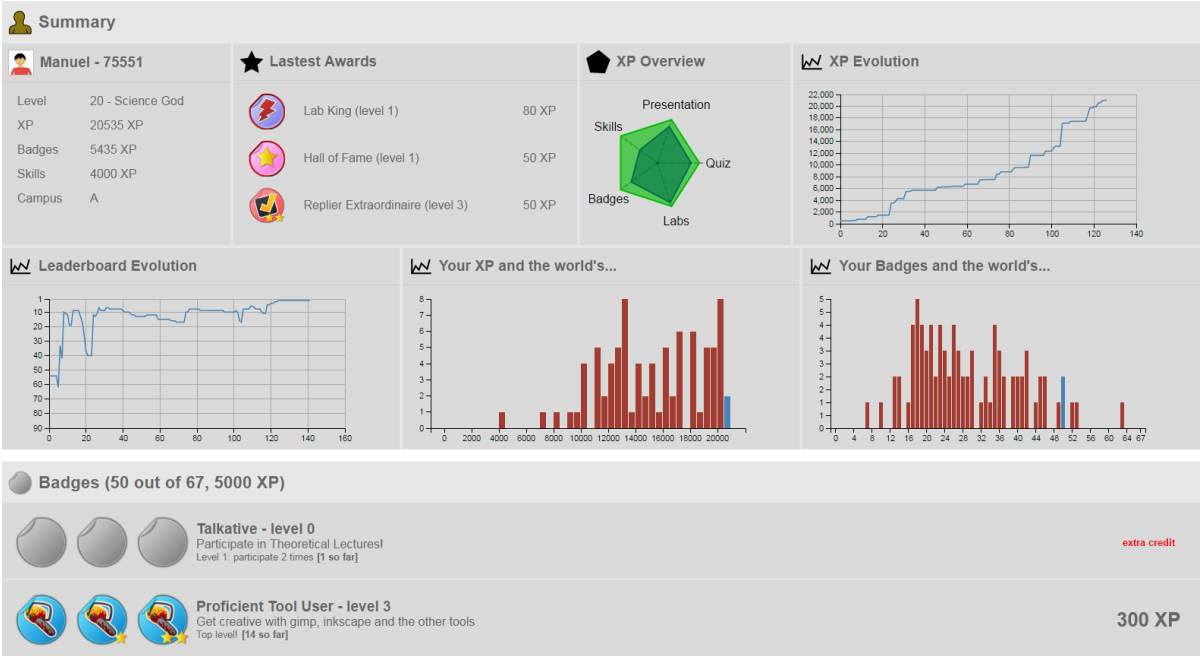


Figure 3.12: Final Profile Page

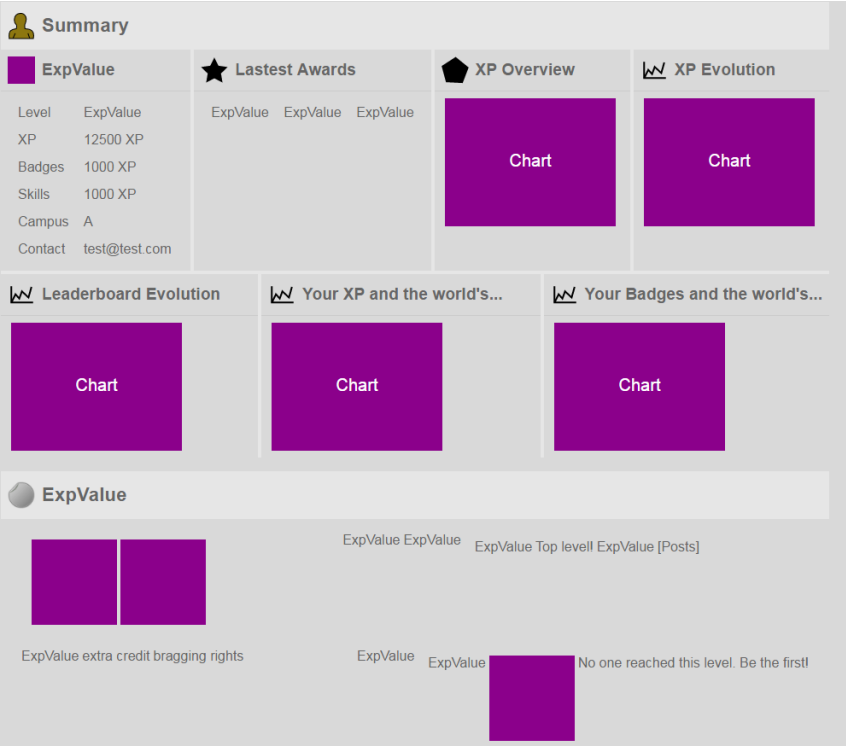


Figure 3.13: Profile Page in the editor

In the functional prototype, a functionality was implemented to test how to add new blocks to the system. Although it was very incomplete, something similar was planned to be implemented for the first version of the system. Using some sort of palette of blocks, the person configuring the profile page would be able to add new blocks to the page. These blocks, would provided by different modules, could then be configured to display the information in different ways. But has is with the leaderboard page, the profile page ended making use of the views to provide different layouts of the system to different people.

3.5.8.3 Side View

The side view is another module that uses the views, it inserts a block on the side of the pages of the leaderboard and profile. It serves the purpose of displaying the information of the student, no matter what page he is viewing. It can be configured has the profile page, since it also depends on the views, but due to the limitation of size it does not fit too much information. This configuration of blocks is very important because if we discovered for example that the users like to compare their awards with others, we could easily add a block that shows awards in a more succinct way and it could be used by as an easier way for the student to compare their awards.

3.5.8.4 Awards and Overview

There are also two modules that provide some information on how the course it going. The Awards module provides some templates and a page in which it is possible to see all the awarded XP since the beginning of the course. This module is also what provides the template for the award list mentioned as part of the main template for the profile page, in which the students can see a list of all activities that earned them some XP.

The Overview module creates a page where it is possible for the professors to define a view to display statistic related to the course, and how things are going. There are some templates available

to be included in this view. One which display the most and least active users, and the other displays all the Skills awarded to students. Other tools to monitor the course can be easily added in the future. One of these tools could be able to analyze the participation in the quests of PCM. The tool could be able to show who first got to a certain level, who has participated/not participated and detect cases like consecutive posts which are not allowed in order to give a chance for everyone to participate.

3.5.9 Other Modules

Other modules were also created for the system. These modules represent other parts of the system, that implement game elements. They are created as different modules to allow to enable/disable when necessary. This makes it easy to use these game elements in other courses, or deactivate them if they are no longer needed.

3.5.9.1 Quests

The Quest is a game element applied in the gamification of the course where the students have a problem and use the tools and subjects taught in classes to find the way to unlock the next challenge. It promotes cooperation between students and rewards all participants involved with experience points.

In the start there was some doubt if the Quests module would be implemented, but it did get implemented. This module allows for the professor to create the Quests for the students, it enforces some of the rules and it keeps track of their progress, which can in the future be displayed in a view.

In previous years, the pages were created using simple HTML pages, and the users were able to skip levels by guessing the keywords of the levels. With the integration of the quest in the system, we can make sure that the player is not able to skip those levels, by allowing to progress the quest only one level at a time. Since the system knows who is logged in, it also possible to track the progress of each individual student (each visit, guess and success).

The module was deployed for the Quest, and some features were suggested by the students. The suggestions were to have the Quest navigation button jump immediately to the last unlocked level and to have a list of all unlocked levels and their keyword. This was suggested because it is hard to keep track of the current level using the forum post. To solve this a link to the last unlocked was added on the page if the student was visiting any other level, making it easier to jump to the last level. This is somewhat better that having the quest navigation button jump immediately the last unlocked level, because someone could want to try to solve the other levels.

With the new module, multiple Quests can be created, one active at each time. The levels are created in the settings page for the Quest module, which also allows to configure other things like the start and end time of the quest, the number of tries a student can do in one hour, and the time between unlocking a level and being able to unlock other one. As the professor writes the code of the page, it updates automatically to reflect the changes and see how the page is looking. It is also possible to upload the resources using the interface to upload the file, not requiring to have access to the server which would be necessary even to create the quest, in the previous years.

Another new feature was the implementation of a mechanism to ask the student for an image to validate the solution of the Quest's level. In one of the quests of 2014/2015 a image was revealed by a student using a method other than intended by the professor, which resulted in the unlock of the next level. With this verification, in which the students upload the 'discovered' image, they will only be able to advance if the image matches pixel by pixel, the solution intended by the professor.

3.5.9.2 Charts

A charts module was created that provides a chart type part for the views, and was not included in the views module, because it is more complicated that the other parts, requiring sometimes the creation of other modules to provide the values for the chart. This chart part has different types of charts, which have very different properties. There are four types of charts provided by the module, bar charts, line charts, star plots and progress bar. All these charts can be seen in figure 3.12. The first two require the registration of a provider that can give the values to populate the charts. This is done this way, because the values displayed are can be really complex, and the expression language is not powerful enough to do it, but could be extended in the future to be. The star plot displays the plot based on properties defined with the values, max and min, which use expressions to get the values. And the progress bar uses the expressions to get these values as well.

Since there is no cache for the views, these charts are computed every time they are required. This is fine for the simpler charts like the star plots and progress bar, but for the more complex bar and line charts, this can degrade the performance of the views, for example in the specific case of PCM where a sparkline is added in the leaderboard page for each student in the leaderboard. This module also defines some providers for these complex charts developed specifically for the use in the course of PCM, using the game elements used in the course. More information regarding to what these charts contain is available in section 4.1.

3.5.9.3 Notifications

The Notifications module was implemented to display notifications (figure 3.14) in the system, when the student receives a award. These notifications simply register functions in the Expression Language, that can then be used to construct different views. In the start of the course of PCM, for the semester of 2015/2016, these notifications were already implemented in the system, but were not as flexible as they are in their current form, which allows to configure all the things displayed in the notification using the views editor. They had to be removed because of the previously explained problem with the database, in which the writes to the database, would make the system crash. With the possible solution for the database, these were implemented again, because they were something that the students liked.

Figure 3.14: Notifications in a View

To construct the notifications using the parts, some styling was necessary to make everything fit the way we wanted, and the Repeat property is used to repeat the notifications which are in an array. When repeated, the notifications then enable a block which 'knows' how to display the type of notification, for example, a badge notification, needs to display the name of the badge and level, while a skill only needs

to display the name, this selection is done with a If property and the type of notification. The 'X' in each of the notifications is used to dismiss the notification, by making requests to the internal API.

3.5.9.4 Game Elements

The game elements of Skills and Badges used in PCM were also specified as their own modules. They provide a link for the data between the data storage (the old script), and the back-end of the system, by registering the available fields related to the game elements in the Data Schema. These are implemented in different modules, in order to make it easy to disable one of them if necessary. Apart from registering the fields in the schema, they also provide default templates for their presentation in the view, for example the Skills module, provides the Skill Tree presented in the profile page of the student. They also register custom functions in the expression language, which allow to make the display in the views more efficient.

3.5.10 How the Views Work

To solve the problem of the configuration of the views, which required a interface to edit the views, we started to develop a new version of that part of the system. As said before, the main problem with the configuration was the representation of the view. With the old structure, there was no representation for an "edit" version of the view, or the data of the view to be displayed to the user. This was due to the rapid development of the system to meet the deadline of the start of the semester with a minimal working version. This version allowed to specify in code the different parts of the view, but the part of editing the views in a editor was neglected, which ended up creating this problem.

It was possible to define different views for different users but their creation needed to be all using PHP code. It also allowed the usage of fields from the database but was very limited. To create a view, a PHP script needed to register it in the tree format discussed before using arrays. This was clearly not what was intended for the system. Furthermore, to implement some parts of the system, like the badge list, which is used by the gamification of PCM to show the user the badges they have achieved, needed to be fully implemented in code as well. This parts used some components that would take the whole page that was being displayed, and detect that the badge list should be inserted there, so it would generate the badge list and insert in that part of the page.

3.5.10.1 First Attempt

To move on from this implementation that did not allow much flexibility, a new structure was created that could be easily used with angular to create the view editor. A new editor and front-end rendered where created, using the new data structure. The parts were rendered as AngularJS directives, which allow to compile custom HTML tags into other tags, essentially extending the HTML. Using these directives allows to make the render of each part separate from each other, allowing to create new parts for the views easily.

Before the implementation of the expression language, the parts of the page used a simple system which created using the data schema to show all the fields available in the database. This system allowed one of these fields, with a format string, to display the result in the different parts, which was not a proper solution, because it did not allow for the necessary expressiveness required to display some of the text in the system. At this point, the expression language was created to configure each of these parts, which made it possible to move away from the components used to insert content in the middle of the pages, and give a finer control to what is display to the administrator, instead of the developer of the module. These components, however, were the base for the creation of templates which are basically

groups of parts predefined by the modules or created by the administrators, which can be inserted in the pages. These have the purpose of making the creation of the page easier, but since they are no longer made in code, they can be changed easily using the editor too.

This first approach to the problem was a very good start, but there were a few problems. Since each part available for the user was its own directive, a common directive was declared that could mutate into any type of part and provided the basic features that are common among all the parts. This additional mutation creates an overhead, because of the compilation of the elements in AngularJS, which slowed down the render of the page. The main problem was using the directives in first place, and having a AngularJS compile them into other HTML which takes time. It worked great for small pages, but slower when there are a lots of elements in the page, for example the profile page, which has lots of images in the badge list. Another problem was the complexity of the code to extend with new types. As the features grew, the code in the directives became very complex, especially with the use of the common directive, which is something AngularJS does not support very well.

3.5.10.2 Solution

In order to finally solve this problem, a new renderer and editor were created in pure JavaScript, with some help from AngularJS. This renderer is defined as an AngularJS service, which allows to create objects with functions available for the whole application. Since this code, that actually transforms the data structures holding the parts into the HTML, was implemented in pure JavaScript without using the directives, it runs very fast, rendering a large page must faster than the other solution. However, there is an overhead with the database (discussed in section 3.5.2) that slows the loading of the page on the server side, taking a few seconds sometimes to load a page, even though the renderer takes a few hundred milliseconds to render large pages.

This new service has a few functions, registering new part types, requesting a view for editing or display, getting registered types and some other functions used by the part types to build elements and assist in their creation. Each part type is registered as JavaScript object with a few functions, to build and do other operations related with the part. Using these JavaScript objects, allows to easily use parts of one type into another type, by looking up the registered type, and directly using the necessary functions. For example, the part type “image”, which displays an image, uses almost all the same code as the part “value”, which displays text. The only difference between the two, is the element that is built, an “img” versus a “span” HTML tag.

The layout editor, provides a way of editing the layout of the parts, adding, moving and removing new parts. It is also possible to save the parts as templates to use in other views. The parts are configured using the Part Settings for each part, which has different properties for each part type, and some properties. These common properties are used to configure the parts for example to repeat them in order to create a list. An example of this is presented in figure 3.15 where we can see the Repeat property defined for the leaderboard page to create the table with the students. The documentation on all parts and properties/fields is available in Appendix B and also in the documentation provided with the system.

3.5.11 Expression Language

As explained before, the Expression Language allows the administrator of the course to configure the views of the system. The main purpose of the language was to have an easy way to access the fields of the system, do operations with their values, and display them in the different parts of the views. They provide a better alternative to the old way of accessing fields using the views, which could only access

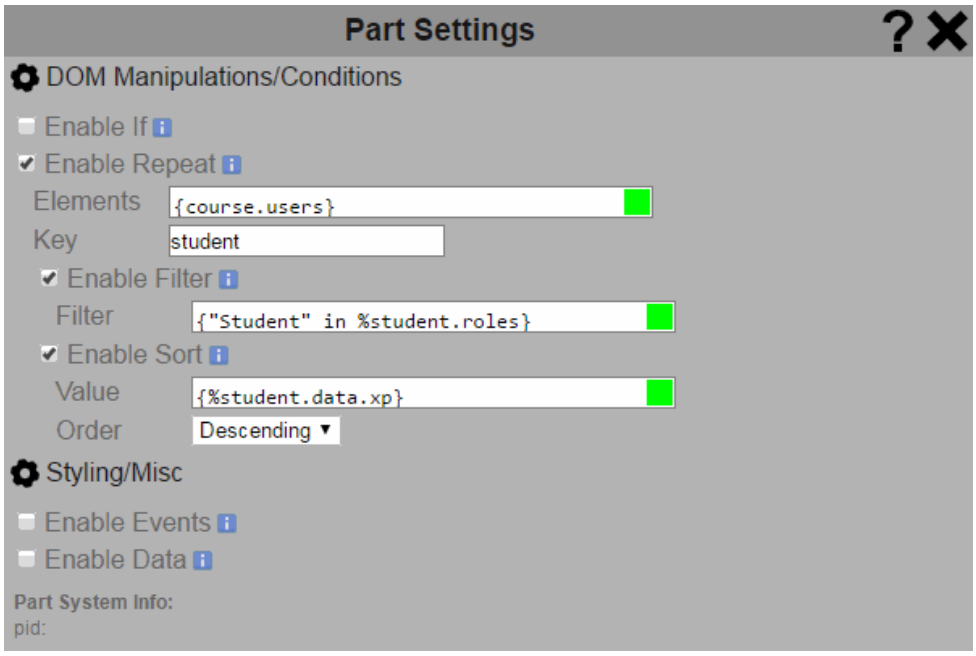


Figure 3.15: Part Settings for Leaderboard table row.

one field and format it, according to a format string similar to the formatted print functions available in the different programming languages. Some of the basic functionalities were already explain in a previews section, but here we give an in depth look on the reasoning behind the syntax of the language.

The first thing required to develop the language was to know all the possible uses for the language. In this case, it is going used in the middle of text to be displayed by the views. This restricts the way it should work, and everything that is typed as an expression is evaluated as a value or text. There is no need for a print function to generate output of the language, because everything that is evaluated is in fact the output. So a hello world example in the expression language would simple be the expression 'Hello World'. To evaluate anything we use curly brackets '{}' which are then compiled and evaluated when the page is created.

For example, to display the sum of two numbers with this language, we would use '{**number+number**}', for example, '{**1+1**}'. This when compiled and evaluated would result in the output '2'. This is the simplest case of using the expressions, which allow for numerous other operators to be used. These operators include the addition +, subtraction +, multiplication *, division / and modulo %. It also provides logic and bitwise operators, and && (bitwise &), or ||(bitwise |), negation ! (bitwise ~) and bitwise xor ^. The common logic comparison operators are also available in the language.

It is also possible to call functions using the expressions. To do this we use the '$' sign, which represents that we are calling a functions. One example of this is '{**$time()**}', which would call the function time and return the current time. These functions are defined by the modules, which makes it extensible. The list of available pre-defined functions can be found in the documentation of provided with the system.

With this functionality its possible to access the database using functions, but it is not practical, because we either need a function to get the value of fields, or have different functions for different fields. To make it easier to access the fields, we use any word that is inside the curly brackets, and does not have a symbol preceding it, as a path in the database. If we wanted to access all the users in the system, we would simply use '{**users**}', which is evaluated as the map that contains the users. To access a value inside the map, we can use rectangular brackets, for example, '{**users[12345]**}' which would result in the object of the user with id 12345. Finally, to access a field of that object, we can

use '{**users[12345].name**}', in which the '.' (dot), is used as a path separator for the object, this would access the name of the user with id 12345. All these fields need to be defined using the data schema discussed before, which provides a mechanism to protect the data that should not be accessible by the views.

This functionality of accessing the database was not always like this. In an early version, to make it easier to implement the compiler, it used a different syntax that matched the field entirely, similar to the simple system with just one field. This simplification was created because in reality the field that is being accessed in the data schema is 'users.user.name' for the example used previously. This would be the field used by the other system. However it needs a context for with user it wants the name. In the simple system with a single field, that context would be provided with a drop-down menu in the editor, but for the expressions it needed to be in the expression so the use of the rectangular brackets would specify the context and the expression was '{**users.user[12345].name**}'. This is weird to look at if you have programming experience with languages that the brackets operator to access arrays and maps, because it barely adds any information to what we already know, that inside the map of users where is a user. So this syntax was simplified.

The language also has the concept of variables. These are defined by the views using the Data property of each part or by repeating an element using the Repeat property that was exemplified in a previous section. There are also a few variables. The 'course' variable has the course id for which the view is being displayed and the 'user' and 'viewer' variables contain the id of the user for which the page belongs to (in the case the page is a profile page, for example), and the id of the user viewing the page, respectively. To use this variables in the expression language we use a '%' sign preceding the variable name, which for '{**%course**}', would result in the expression returning the value of the variable 'course'. These variables can have complex values like maps, arrays and objects, which for the purpose of the variable they are returned as continuations from the data schema or the function that provides them. Using these continuations is a good way to make the views faster, as they cache the value and the schema path that the value belongs to, allowing the access to the children of the held value to be much faster. For example, it is better to save the user object in a variable, and then accessing its children multiple times, than accessing its children multiple times directly from a full path.

In order to make all these features of the language work, it was necessary to create a evaluator for the language. To do this a parser generator was needed with the capability of running in PHP, and also should not be that complicated to create. A few options were considered, but the decision was to go with Jison (available at github.com/zaach/jison), which has an API similar to Bison, which is used in the Compilers course at IST. Having the knowledge of how to define a grammar for this parser generator made it easier to develop the grammar used to evaluate the expressions. This parser generator was developed to generate parsers in JavaScript, but there is a port which generates the parser in PHP as well. At first the result of parsing a expression with the parser, resulted in the expression directly, but in order to have more power over the whole system, a iteration was created which generates an abstract syntax tree, which contains the nodes to be evaluated at a latter time. This also opens the possibility, in the future, for caching this tree to make the views faster, even though the parser only takes a few milliseconds for a fairly complex page like the profile page.

All the power of the expressions comes at a cost for the usability of the system, as it forces the user to understand the expression language to configure some the parts of the views. Maybe this could have been avoided by using some other system, like a visual programming language, but that would take much more time to implement. This cost in usability can be somehow minimized by providing great documentation on the system, on how to use it properly. However, not everyone consults the documentation, and should not be expected to know every field available in the data schema. For this reason, a auto-complete function(shown in figure 3.16) was implemented for database paths in used

in the expression input boxes provided for editing expression on the views. Since the parser generator used was developed to work for JavaScript, we do not need to define the grammar for other parser, and can build the auto-complete feature with the same grammar used in the PHP parser generator. It only works when completing a path, and not for completing other things like the variables or operators.

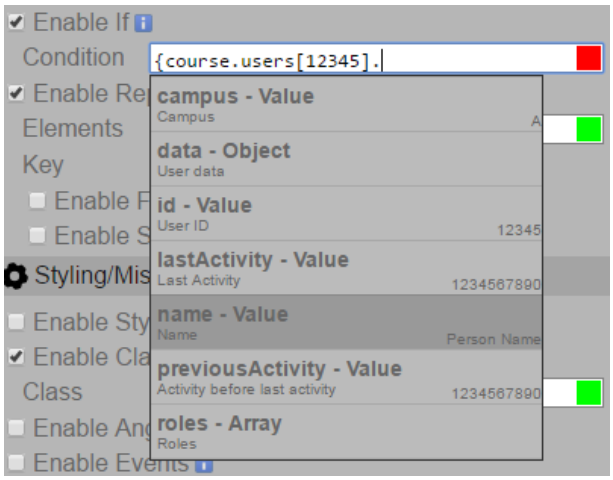


Figure 3.16: Auto-complete Feature for Expressions

This auto complete feature is very helpful to speed up the creation of expressions, even though it is not very advanced, as it stops working if there is an error in the expression, before the path we are trying to use the feature one. This could be corrected to work in this way, like many IDEs do, but would take some time to figure out exactly how, because the expression could serious errors and we could be suggesting a completely wrong thing to the user, so its safer to not suggest anything at all. One interesting feature would also be the ability to suggest from variables containing continuations, but it would require to request the evaluation of the variable for that view, and it gets complicated for continuations that come from functions and not the data schema. Documentation for the Expression Language, and how to register new functions is available in Appendix B and also in the documentation provided with the system.

3.5.11.1 Advanced Examples

With all the concepts of the language explained before, we can now construct more powerful expressions. For example, to display the XP of a user with id 12345 in the format like '1500 XP', we would use the following expression '{**course.users[12345].data.xp**} **XP**'. Note that in the expression, the ' XP' string is outside the curly brackets, because it does not need to be evaluated. As stated before, the language has pre-defined functions, one of these functions is the 'time' function, which returns an unix epoch timestamp in seconds. We can use this for example to show how many seconds ago the first award was given to a student. To do this, we use the following expression **'Awarded** {**$time() - course.users[12345].data.awards[0].time**} **seconds ago'**, this would output something similar to 'Awarded 100 seconds ago'. It begins by evaluating the time function that returns the current time, to which we then subtract the award time, for the first award, in this case the award with index number 0 in the array of awards for the user in its data. These are two examples of how to use strings mixed in the expressions, which contain functions, data access and operators.

Expressions can also be used as key for a data access, for example, '{**some.path[**{**123 + $func()**}**]**}'. This example would not actually return anything in the SmartBoards, and is just used here to display the power of the expressions. In this case, the 'some.path' would be an array or map, which then would

be indexed by the result of the expression '{**123 + $func()**}'. One example of the use of this, would be for example to select a user based on some criteria. Let say that the function 'latestStudentWork' returned the id of the user how did the most recent work for the course. We could have something show up in a page, like the name of the user and XP. To do this, we would write the following expression: '{**course.users[$latestStudentWork()].name**} **did the most recent work for the course. He/She now has** {**course.users[$latestStudentWork()].data.xp**} **XP.**'. Of course this could be improved by applying a styling and including images or actually having a function that returns the id or their latest work (or award) so that it could be displayed.

3.5.11.2 Extending the Language

It is possible to extend the Language with the functions. As referred before, these are registered using modules, and can do anything in the back-end. It is easy to define them, because they are simple lambda functions registered in PHP which are then passed to the expression evaluator. These functions can be defined with any number of arguments because the evaluator will do an automatic match with the registered functions to check if the number of parameters that the function was called matches the number of parameters accepted by the function. In a previous section we suggested for example mapping the mathematics function 'sin' from PHP into the expression language. This could be easily done with a single line of code. Other examples of the functions is a function that is declared by the badges module, in order to cache the badges and speedup the loading of the badges in a student's profile. This function is called by the badge list, to cache the badges that all users have, which can then be accessed by a function as well, when the information of the badges is needed. This speeds up the loading because the badges do not have to be loaded from the database again.

This is the only current way to extend the language, apart from editing the code of the evaluator directly. By editing slightly the way the evaluator works, maybe it would be possible to define new evaluators for the expressions or extend the language in other ways. One example of such extension is discussed in the potential of the expression language, in section 4.2.2.

3.5.12 Multiple Courses

After having a single course working, it was time to start working on supporting multiple courses. This feature was first discussed with the professor, and supposed to work with only one course active at a time. With further discussion, it was agreed that having multiple courses, would make the system better, not requiring to have multiple instances of the system running to support different courses. However, this change required to do some modifications in the interface and internal API, which requests the pages. Both the user interface and the API had to take into account the course which the user was visiting, which meant changing the URLs and creating new pages to link the system together.

It was also necessary to create a way to load the resources of the course's modules, which need to be loaded into the angular application in "run time". To do this loading an angular module called ocLazyLoad is used, which when provided with a list of resources can load them even in run time, not being necessary to load the page. One other problem that had to be taken into account was the initialization of the modules when the courses change. For example, if we load a course with a module, we need to make sure that when we load the module on the other course, it will perform any initialization required for the module to run in that course. That is the case of the Side View, it was required to receive the event of state change to check which course was active at the time, to request the side view to be displayed for the user.

3.5.13 Minor features

The system ended up having many complex features that really define the system, however, there are some minor features that were also implemented, which are not much significant but are necessary to have. One of these small features, is a script that is used to download the photos of the students from the Fenix system to be displayed in the leaderboard, for example. The Fenix system has an easily accessible URL that can be iterated using the usernames of the students to get their photos. This script, uses this URL and the cookies of an authenticated user (because some images are not public) to download the photos, so that even if the session (in the Fenix system) expires for the person viewing the leaderboard, the images would still show. In the future, the system could be improved to allow the upload of user photos in order to update them.

To get the usernames for the users it is also necessary to run a script, because the old python script did not use usernames as the key to manage the information, but these usernames are necessary to identify the user in the Fenix system. The script uses the Fenix page of a course, where there is a list of enrolled students and their respective ids and usernames, to associate them in the database. There is also a simple invitation system, that was created that allows the administrators of the system to invite new people to have access to the SmartBoards. When the administrator wants to invite someone, an invite should be created with their id and username in Fenix, and upon logging this person logging in, the invite is destroyed and they can see everything in the SmartBoards.

Near the end of the development, a simple external API was developed to serve information relative to system and each course in specific. This API basically dumps the entire contents of the files in JSON format. And also can be used to update the keys in the files. The API is protected by a key, as expected, one for the core of the system, and one for each of the courses. This was developed with the intent of allowing other external applications to interact with the system. The implementation allows to define functions easily for any HTTP method, with a few lines of code, which makes the API easy to extend if needed. Everything that is saved at the moment in the system, can be changed using the API, except the API keys, however, there are not many checks in place to verify if the information being updates will make the system stop working. To authorize the request to the API, the server accepts the HTTP Authorization header, which is used traditionally to authorize used to view a page in a certain web site, but it is used by many REST APIs as their authorization mechanism. In order to make it easier to use, it also provides this authorization simply by sending the key as a parameter of the HTTP request, in either a POST or GET request. There are a few functions pre-defined in the system, for both the core part of the system, and for the courses. These functions can get all the data the system has, and update anything, by sending an update key when requesting the function. More information is available in the documentation included in Appendix B.

In the future a better API should be implemented, because even though this general purpose function updates the data directly and allows for a larger control of the data stored in the system, this is not a proper solution for a system like this. Functions that serve specific purpose for example setting a course name, or adding a user, should be implemented because they limit the control of the API, enforce proper rules of consistency in the database and can increase the security of the system.

3.5.14 Installation and Documentation

Even though the system has many functionalities its installation is not difficult. A guide was created on how to install the system based to run with the old legacy files. It has a few steps that involve copying files, running scripts or configuring a few files. One thing that is required to run an instance of the system is to create a FenixEdu application in Fenix. This provides the login as explained in a previous section.

There is also the requirement of having the npm (a JavaScript package manager) installed with the 'less' and 'jison' packages. The 'less' package is used to generate the CSS files, from their source files, which are written in LESS to take advantage of the advanced features it provides. The 'jison' package is the parser generator, used to generate the parser for the Expression language grammar used by the views.

After having all these requirements sorted out, and the SmartBoards installed, the information of the students and teachers should be loaded into the system, their usernames updated, using the script, and the photos downloaded. After this, it is required to setup the views for all pages (for example, Leaderboard and Profile). This setup is only necessary for one course if when creating a new course, the old one is duplicated. In order to make all this faster, it is possible to use the predefined templates defined by the modules, which make for a great start in the development of more elaborate views.

To assist with all this installation and configuration it is necessary to have some documentation. The documentation provided helps to setup the course and the views. There is also documentation on how to extend the system, with examples on how to do specific things, like registering a new type of chart, or how to access user data from a module. This is important to make it easy for someone who is not familiar with the system, to pick up an idea of how the system work, and be able to develop a module that integrates with it.

There is also documentation available on the data schema for each course, shown in figure 3.17. This provides an easy way to check what information is available to use in the configuration of the views, since it will show all the fields registered to the data schema of a course, and is updated if a module is enable or disabled. The developers can also consult this to see what the other modules are storing in the database, and have access to it with the module they are developing. The page provides a search capability that makes it easier to find the fields we want. The search is powerful as it can match partial words in the actual field name, the path of the field, or any word in the description.

Finally, there is information related to the External API, which was discussed in a previous section. This documentation has all the steps needed to connect an external application to request or update information of the system. All the errors are documented with the cause for them, and the API functions that are available have a brief description of that can be accessed with them, as well as which parameters they require.

3.6 Effects on Motivation and Engagement

The new features that allow for the differentiation of views for the different students were introduced to allow the study of the effect on students' motivation and engagement. Using the research done by the various studies presented in Related Work (chapter 2) it is possible in the future to use all the features provided by this new system to study the effects of the different types of leaderboard and functionalities that were appreciated by the users in those scenarios of gamification. Features like the implementation of notifications of badges, that can possibly have the effect of instant gratification on the users.

Docs | Schema | API

Schema for Course: 0

Toggle All

users - ***Map*** Field Expression

Description: All Users

Copy to clipboard

user - ***Object*** Field Expression

Description: User

name - ***Field*** Field Expression

Description: Name

Example: **Person Name**

email - ***Field*** Field Expression

Description: Email

Example: **test@test.com**

username - ***Field*** Field Expression

Description: Username

Example: **ist112345**

course - ***Object*** Field Expression

Description: Course

moduleData - ***Object*** Field Expression

Description: All Module Data

Figure 3.17: Data Schema for a course

Chapter 4

The PCM Case Study and the Potential of SmartBoards

The first instance of the system was deployed for the course of PCM. In the following sections we discuss the specific configuration for PCM and the potential of the system.

4.1 PCM

In the case of PCM we used the Leaderboard, Personal Profile pages and the other games elements that were developed specifically to be used by the course. The Leaderboard and Personal Profile pages are still the main pages for the students, but can be configured by the professors to change the way they are displayed in order to improve the students' motivation and engagement. PCM uses the Badges, Skills, Awards modules to provide the basic game elements for the gamification. The Badges game element is used in the course to give a reward to the users when they accomplish a task, from posting on the forum and submitting their assignments, to participating in the classroom. Each of these rewards have a XP value associated with them, which gives some points towards the final grade in the course. Some of the work given to the students is provided through the game element of Skills, which specifies a task that needs to be completed for each skill, that the user can unlock to earn some more XP.

In terms of configurations, in the leaderboard page of the old system, students could only see their position ordered by XP, but the new leaderboard allows them to order by every column and can be configured to present all students or just some students that are in a certain range of the student consulting the board in terms of position on the overall board or badges collected, for example. The default template has the same columns as the old system, but it can be changed by the professors to display the information that is most relevant to the students. This easy configuration of what is displayed in the page introduces the possibility for displaying other components, for example sparklines[1] for the evolution of XP, or some component that shows most recent/unique achievements. This spark line is one of the new things that was added to the page. The old leaderboard and new leaderboard can be seen side by side in figure 4.1.

The Personal Profile page was also changed and its configured as the new entry point of the system for the students and can be configured by the professors to display the relevant information to the users. The default view for both the students' and professors' profiles starts with a "character sheet" that displays the basic information of the user, like their name and photo. On the students' profile it also displays their level, XP, and number of badges and skills earned on this character sheet. The students'

[1] https://en.wikipedia.org/wiki/Sparkline

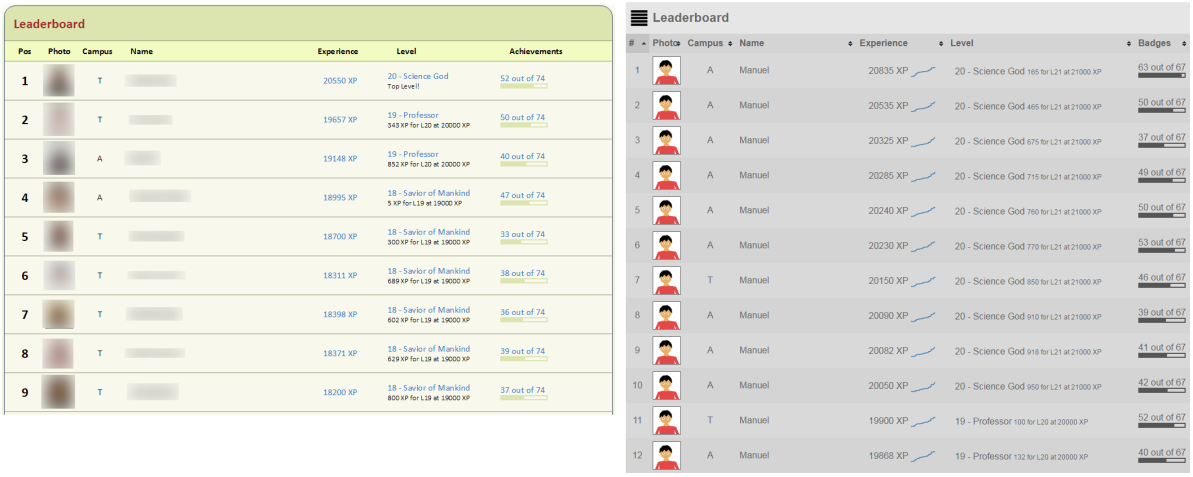


Figure 4.1: Old Leaderboard and New Leaderboard side by side

profile also displays line charts with the evolution of XP and leaderboard position, and bar charts to see the distribution of XP and Badges. It also has the Badges, Skill Tree and XP List that the old system also presents. They show the information related with the game elements, and provide the complete view of the activities performed by the students. Some new features implemented in the page were notifications for completion of Badges/Skills and recent grades on Quizzes/Labs, a chart comparing the different components necessary for the grade in the course and other minor features.The old and new profile pages can be seen side by side in figure 4.2.

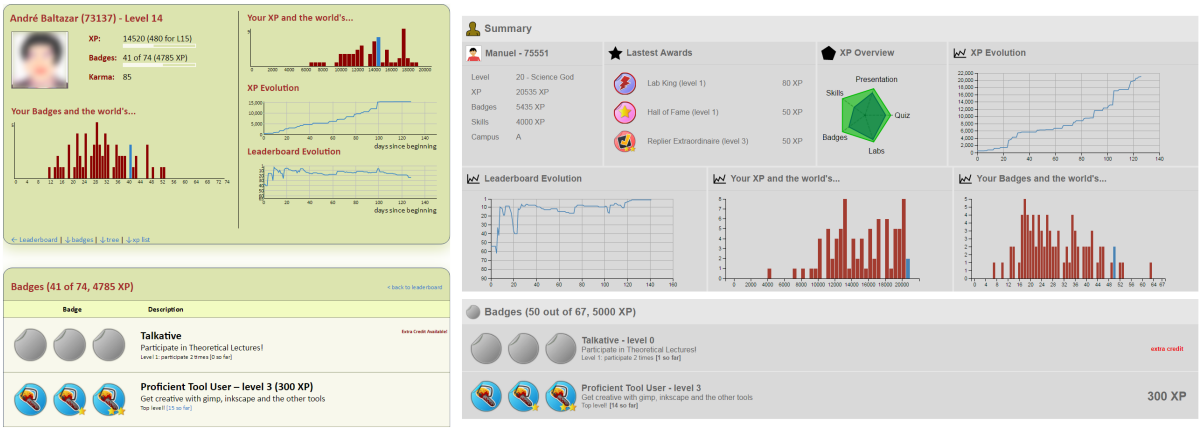


Figure 4.2: Old and New Profile Pages side by side

For the previous years, the old script was used to generate these pages but in a static form, and since the script was hundreds of lines long, it was also very difficult to modify them. However, with the implementation of the SmartBoards, this task is made much simpler by using the views, which can represent all the things that used to be in the static pages, and provides the capability of editing them easily. To load the information from the old script into the database of the system, a PHP script was developed, that should be run after the python script, to read all this information. This script is the less flexible part of the SmartBoards system as a whole, because everything that is in the script is specific to the course of PCM. It reads from the different files produced by the script and knows where it needs to place the information so that it can be accessed and displayed by the different modules. Looking back, one idea to improve on this script while maintaining the idea of customization of the SmartBoards system, would be to migrate this script into a module that would be specifically implemented for this course, with some configuration options available in the settings page to control the things that need to

be loaded in the system.

Implementing all the old pages in the system was not difficult, but to do this in the views, it requires some files to be edited, instead of being able to use only the properties available in the views. This is because these pages are advanced in terms of styling and interaction. All the styling of the pages were defined as CSS classes in files registered by modules, which makes it more complicated to change the styling than going to the properties of the different parts but not that much more difficult. Even though changing the style might be somewhat difficult, changing the content of the page is easy using the views editor, if the person configuring the has some experience with the Expressions. This is an advantage in comparison with the old system, which to change anything, the script needed to be edited and all the pages generated again.

In terms of student feedback, they seemed to like the new system. They reported a lot of bugs, and gave suggestions for new features and improvements. The bugs reported were resolved, and some of the suggestions were implemented, like having the link to the post of a skill, show in the skill tree. Features like these are good to navigate easily between the SmartBoards, and the forum. One suggestion for the side view, was to show the ranks and xp earned since last login, this could be implemented in a module for this purpose. Other suggestions were related with the charts. These charts did not improve very much from the charts of the old system, but the students suggested for example allowing to move the mouse over the line in the line chart to see the value at that point.

4.2 Potential of the SmartBoards

In this section we explore the potential of the SmartBoards system. What it is capable to display, and the ways it can be extended.

4.2.1 Views

The Views allow the administrators to really extend the system in any way they want. It is possible to do some extreme examples of views that can be configured by them. In this section we give some of these examples what could impact the students, have functionality that allows to monitor the course or implement some other mechanic.

One example that both could have an impact on the student and also can be used to monitor the course, is to display which students are on a path to failure. The figure 4.3 uses the template of the Leaderboard to create a list of students that do not have a very good amount of experience points relating to the days left to the course. In the example we simply divide the total days by the number of elapsed days and multiply by the XP to check if they can get a number greater than 10000 (minimum required). Of course this is not a great formula to compute how is going to fail and how is not, but a function with some other kind of inputs could be used to greatly improve such estimation. An example of this is the related work of the course of PCM where the students were put into categories based on their performance. Some sort of machine learning component could be created to do this automatically. Other information could be displayed, like number of quizzes attended (or missed), has grade on the presentation of PCM, etc. This example could be used by the professors to talk to these students in order to give them some guidance, or with the use of this integrated in the side view, we could show an alert that the student should start working more in order not to fail.

The leaderboard is other thing than has the potential to change. As discussed in the related work, there was some research about different types of leaderboards. For example, the leaderboard in figure 4.4 shows the Top 10 users, and then in the buttom, the student that is currently viewing the leaderboard,

Could be failing the course! (40 days left out of 160 total)

#	Photo	Campus	Name	Experience	Level	Badges
1		A	Francisco	4025 XP	4 - Taking your first Steps 975 for L5 at 5000 XP	7 out of 67
2		A	Maria	7473 XP	7 - Attentive Pupil 527 for L8 at 8000 XP	10 out of 67

Figure 4.3: Students that could be failing the course

in this case this student is in rank 33, as seen in the figure. This is just one example of the types of leaderboard discussed, and implementing other types would be just a matter of selecting which rows to display in the table. To implement this, we simply use the If property in the row of the table, which allows us to hide the row based on a Expression. Another example of a leaderboard could be for example, showing the students that are on top 3, plus the students that are in a 500 XP range, as shown in figure 4.5. This is done, by changing just the Expression that decides to show, or not to show, the row of the student.

Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
1		A	[redacted]	20835 XP	20 - Science God 165 for L21 at 21000 XP	63 out of 67
2		A	[redacted]	20535 XP	20 - Science God 465 for L21 at 21000 XP	50 out of 67
3		A	[redacted]	20325 XP	20 - Science God 675 for L21 at 21000 XP	37 out of 67
4		A	[redacted]	20285 XP	20 - Science God 715 for L21 at 21000 XP	49 out of 67
5		A	[redacted]	20240 XP	20 - Science God 760 for L21 at 21000 XP	50 out of 67
6		A	[redacted]	20230 XP	20 - Science God 770 for L21 at 21000 XP	53 out of 67
7		T	[redacted]	20150 XP	20 - Science God 850 for L21 at 21000 XP	46 out of 67
8		A	[redacted]	20090 XP	20 - Science God 910 for L21 at 21000 XP	39 out of 67
9		A	[redacted]	20082 XP	20 - Science God 918 for L21 at 21000 XP	41 out of 67
10		A	[redacted]	20050 XP	20 - Science God 950 for L21 at 21000 XP	42 out of 67
33		A	[redacted]	17020 XP	17 - Exalted Savant 980 for L18 at 18000 XP	35 out of 67

Figure 4.4: Leaderboard Top 10 with User

Another example of the potential of the views, is a gallery of work submitted by a student, as seen in figure (4.6). This is an example of something that could be implemented in the system, but it is a lot more complex to implement that the other examples, because it would require some code in the back-end, to make the comments system work, to allow to register a new type of part in the view, and to be able to input the comment. Even though it would be more complex, everything is already available in the system to make it work, except the input box in the views. The comment of the users would be saved in the database, and then to produce the list of comments the Repeat property would be used to show all of them. To post the comment, we have the internal API, that is accessible by the JavaScript, which would only need to do a request to save the comment in the database. The only thing missing for the views

Leaderboard

#	Photo	Campus	Name	Experience	Level	Badges
1		A		20835 XP	20 - Science God 165 for L21 at 21000 XP	63 out of 67
2		A		20535 XP	20 - Science God 465 for L21 at 21000 XP	50 out of 67
3		A		20325 XP	20 - Science God 675 for L21 at 21000 XP	37 out of 67
28		T		17355 XP	17 - Exalted Savant 645 for L18 at 18000 XP	36 out of 67
29		A		17207 XP	17 - Exalted Savant 793 for L18 at 18000 XP	28 out of 67
30		A		17192 XP	17 - Exalted Savant 808 for L18 at 18000 XP	29 out of 67
31		T		17132 XP	17 - Exalted Savant 868 for L18 at 18000 XP	30 out of 67
32		A		17082 XP	17 - Exalted Savant 918 for L18 at 18000 XP	26 out of 67
33		A		17020 XP	17 - Exalted Savant 980 for L18 at 18000 XP	35 out of 67
34		T		16712 XP	16 - Society's Best Hope 288 for L17 at 17000 XP	35 out of 67
35		A		16579 XP	16 - Society's Best Hope 421 for L17 at 17000 XP	24 out of 67

Figure 4.5: Leaderboard Top 3 and 500 XP range

would be the input box, that would require only a few lines of code to be registered as a new part in the system. This example showcases very well what is possible to do with the SmartBoards system, and just a few modifications.

4.2.2 Expressions

The Expression compiler that was created for the views also as some potential for other things. Since it produces nodes that can be accessed by the PHP, other parts of the system can be extended to use the expressions.

One example of this could be the definition of rules to unlock the achievements. As discussed in a previous chapter, the SmartBoards system only displays the information that is processed by the Python script that used to power the course of PCM. With the use of expressions a simple system of if-then to unlock achievements could be implemented. For example, an expression such as '{$tryUnlockBadge(%student, "Post Master", 1, (%student.data.postCount>20))}' (which calls the 'tryUnlockBadge' function) could be run for each student, and if the condition in the last parameter was met, it would unlock the badge Post Master (level 1), for that student.

An extension of the Expression compiler could also be created to allow to define the functions in the system, instead of having to define them on code by editing the module files. This would make the expressions a lot more powerful. Defining functions for things that are common between the views could make defining views easier. This option of extending the compiler, also benefits the example of using the compiler for other things other than the views. With the introduction of loops and function definition, it would be possible to define scripts that run on the system to perform some sort of task. For example, if we wanted to create a file with the name and grade of all students in the system. A simple function to save a file could be implemented in PHP, and then a script using the Expression compiler created that loops for every student and outputs their name and grade to the file.

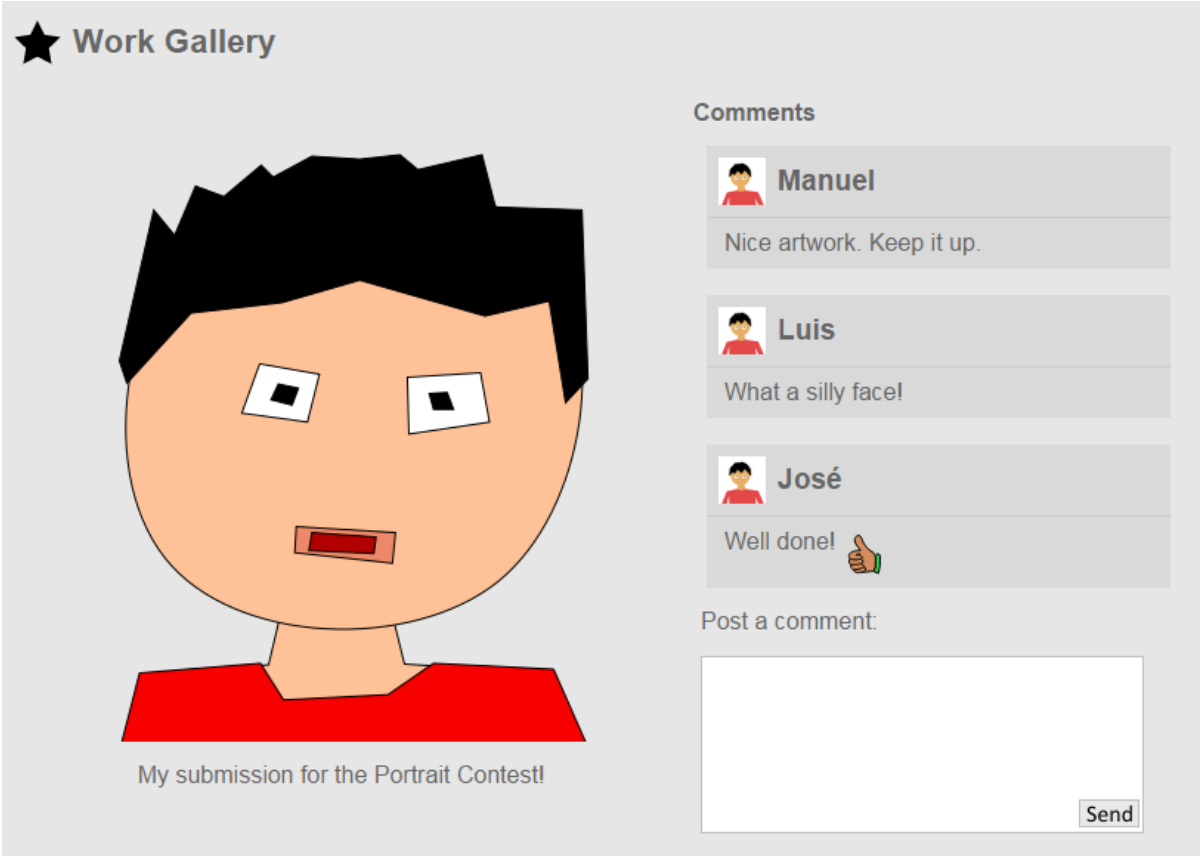


Figure 4.6: Work gallery with comments

4.2.3 Modules

The implementation of modules into the system allows for a great extensibility, with the advantage of being able to turn that part of the system on/off without having to change code. This adds huge potential to the system, for example, being able to add pages to the SmartBoards system, when they are needed. Let's say there is some assignment that needs to be done, which requires some setup, a module could be created to provide a tutorial for that setup. This could be enabled only before the assignment to display that information.

Following this idea of a tutorial, the Quest module, which has various levels and an editor to configure them, could be easily turned into a module that allows the professors to define tutorials, all within the system. Only few changes would be required, removing the unlock mechanism, displaying all the different pages of the tutorial and probably allowing multiple tutorials to be active at a time.

Another example of a module is the implementation of the Hall of Fame, which is something that PCM has in the end of the course, which displays the best work done by all the students. This could be implemented as a module, and enabled only when needed.

Chapter 5

Evaluation

In order to evaluate the system, it was deployed for the course of PCM during the second semester of the school year 2015/2016. However, this deployment did not give us much information for the evaluation, contributing mainly for fix bugs and polish features that the students wanted. To take in consideration the feedback from the users, we were going to implement a system similar to IST-DSI FeaRS (Feature Request System) used in IST in 2009, which was used to collect feedback, feature requests and bug reports related with system. No such system was implemented, although it could be easily created as a module for the SmartBoards for future research on what the users want for the system. The questionnaires that are used by the professors to collect feedback from the various aspects of PCM were considered as a form of evaluation of the system, however, the students do not have access to the part of the system that would allow them to edit the views which make these questionnaires useless for the evaluation. Thus, a summative user evaluation was conducted with users performing specific tasks, which were measured for completion time, and number of errors.

5.1 Summative Evalutaion

This summative evaluation allowed to measure the usability of the system, by measuring the parameters referred above during the execution of the tasks, by using some questions and with the usage of System Usability Scale (SUS)[21] method, which is a reliable tool to measure the usability of a system. To perform these evaluations a few restrictions were put in place in order to control the scenario of the tests and have more accurate results. The system was setup exactly the same way every time a user had their first contact with it, and no help was provided during the realization of the tasks. The users were allowed to skip the tasks if they could not complete them, and in the case the completion of a particular task was necessary for doing another, the task was setup by the observer with a brief explanation of what was done. The users were allowed to consult all the documentation available during the tests, as well as the views already configured in the system.

Before starting the tests, the system was presented with a brief explanation about the objectives and purpose of the system. After this explanation, an example was created to demonstrate the creation of a view. This example consisted in the creation of a view to show the badges of a user in the system. To accomplish this, a block was created in a view, similar to what is show in figure 5.1, with the header title 'Badges' and a custom image. A image was added inside this block to be repeated, using the Repeat property of the views, to create the badges of the user. For the user to understand this, a brief explanation was given about how the Expression language works, how to get the necessary information from the documentation, and was allowed to ask any questions about it. With this demonstration complete,

the users were allowed to explore the system and documentation as they wished, creating elements, consulting the schema, etc. A questionnaire for user characterization was answered by the user after they explored the system, which then allowed the users to start executing the tasks. This questionnaire asked for the age and gender of the users, as well as their experience with complex applications, which were considered applications that require a little more learning than more common applications, experience with content management systems and with web technologies.

Figure 5.1: Example used to explain the concepts of the Views in the system.

The test consisted of seven tasks that, as a whole, produced a table with the active users in the system, as shown in figure 5.2. For each task a time limit was specified, to ensure that the user would not get stuck and frustrated with the task, however, this could be extended if the user was near the completion of the task with few errors. After each task, the users were asked to rate the difficulty and give some comments about it. The tasks were presented to the users as follows, and a clarification of what was required was provided if necessary:

1. Create a container block with the header text ”Last Activity“ and ”images/awards.svg“ image
2. Create a table with 3 columns and a header row (”Photo“, ”Name“, ”Last Activity“)
3. Turn the table into a list of course users, sorted by last activity, filtered by last activity (show users with activity only)
4. Display the photo of the user in the first column

   Photos of the users are located in the “photos” directory, indexed by username and have “png” extension. Example path: photos/ist173137.png
5. Display the name of the users in the second column
6. Display the last activity of the users in the third column
7. Format the table

   Center the ”Photo“ column, and apply the predefined ”user-image“ CSS class to the photo. Center the ”Name“ column. Style required to center the columns: ’text-align: center’.

The expected times and maximum times for each tasks were as follows:

1. 45 seconds (maximum 90 seconds)
2. 90 seconds (maximum 180 seconds)
3. 120 seconds (maximum 240 seconds)
4. 90 seconds (maximum 180 seconds)
5. 60 seconds (maximum 120 seconds)

Figure 5.2: Expected result upon completion of the tasks in the evaluation.

6. 60 seconds (maximum 120 seconds)
7. 90 seconds (maximum 180 seconds)

These times were based on the difficulty of the task and the times from a pilot test, which was conducted to see if a user would be able to understand the tasks, and to catch some bugs that might occur. After the realization of the tasks, the following questions were asked to the user, which allowed us to collect some more comments about the system:

- Difficulty adding new blocks/tables/images? (1 – Very Hard; 5 – Very Easy)
- Difficulty using Expression Language? (1 – Very Hard; 5 – Very Easy)
- Difficulty using variables in the Expression Language? (1 – Very Hard; 5 – Very Easy)
- Difficulty using fields in the Expression Language? (1 – Very Hard; 5 – Very Easy)
- The documentation was helpful to complete the tasks? (1 – Strongly disagree; 5 – Strongly agree)
- Could you understand the available fields in the schema? (Y/N)
- Difficulty understanding the documentation? (1 – Very Hard; 5 – Very Easy)
- Comments about the system

The evaluation concluded with the realization of the SUS, and a 'thank you' to the user for taking the time to do the evaluation.

5.2 Results

In order to investigate the result of the test, a statistical analysis was done to each of the measured parameters for each task, and the questions asked to the users. For continuous variables, the mean, standard deviation, median and quartiles were calculated. The quartiles were also calculated for ordinal variables, along with the median and mode. Finally for the nominal variables only the mode was calculated. In the cases that it made sense, some percentages were calculated, like the percentage of completion of each task. All the raw data can be found in the appendix chapter A.

The tests were conducted mainly on male users (19 out of 20) with average age of 22.9 years old (standard deviation of 2.92, median 23). The majority (95%) of them have knowledge of more complex applications, that require learning something about the application and are not simply applications that are used by everyone, like e-mail applications or social media. Around half of the users tested (55%) have used a Content Management system, which could have some benefits when using the SmartBoards system. Finally, 80% of the users answered that they have used web technologies like HTML, CSS and JavaScript.

The task completion was very good for tasks 1, 5, 6 and 7, with a completion of 100%, and for task 2 with 95% completion, while for tasks 3 and 4 it was lower, with 60% and 40%, respectively. This was expected due to the difficulty of these tasks, as the task number 3 was the first task that required the users to learn the expression language, and task number 4 used some complex features of the views and expression language. The expected times were inside the confidence intervals (at 0.05 significance) for tasks 2, 5, 6 and 7, and were below the interval for other tasks. This means that the users took more time than expected to complete the tasks. For task 1, this is not a big problem, because if the estimation was two seconds more, that would have hit the interval. However, the estimation for the tasks 3 and 4 which also had problems with the completion were much distant from the estimation. The expected times that are inside the confidence internals are also not very problematic, because with a few seconds more (2 to 17 second in the worst case) to complete each of these the expected times would be greater than the interval. In figure 5.3, the mean times and standard deviation of each task are shown.

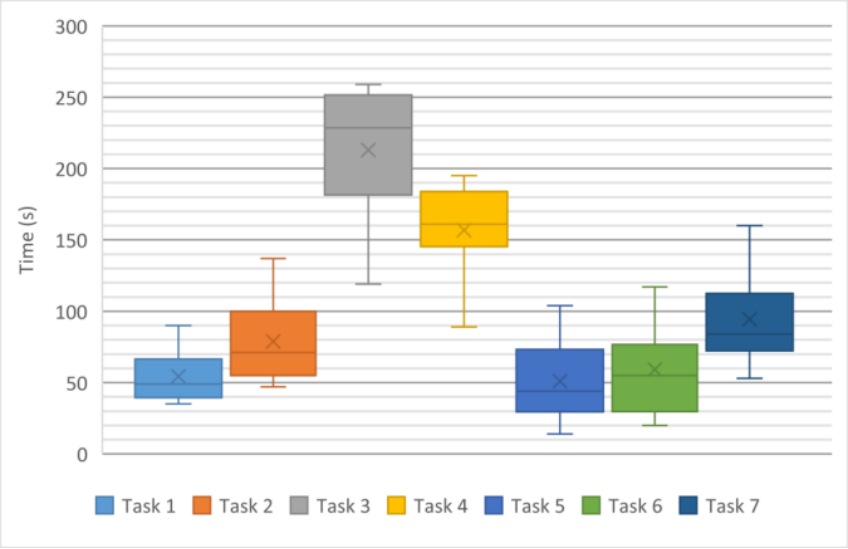


Figure 5.3: Mean task execution time with standard deviations.

In task 1 the users had a mean number of errors of 1, with a standard deviation of 1, the majority of the errors here where due to the way the block are added and them manipulated. To add the block the user needs to start editing the layout of the page, then add the block, and finally deactivate the layout editor again to be able to edit the title of the block. Some people started dragging the block around when trying to access the settings menu for the title. One other common error here was accessing this settings menu, instead of accessing the menu for the title, they would access the menu for the whole block. Even though some users had this problem, they still rated the task very easy, because they understood the problem. The task 2 which was the creation of a table, felt by the users to be a little more complicated, because the first quartile was difficulty 4. The main problems here where how to access the table layout editor, and how to insert the columns, because they did not understand the way layout editor worked.

Task 3 was the first hard task for the students, because it introduced the Expression language. As mentioned before, only 60% completed this task correctly, but even with this number of completions, the users felt that it was not hard, because the rated difficulty of the question is uniform, trending for an easy difficulty (mean of 3, mode of 3, and quartiles at 2, 3 and 3). Some users struggled with the syntax of the language, specially those who commented that did not have much experience with programming languages, so using the language was harder. Other common errors were not understanding how the Repeat, and variables actually worked. There was some struggle with the usage of the database fields. The task 4 which also involved the usage of the expression language, was the most difficult of all to the

users, with a success rate of only 40%. This is due to the fact that the task required the users to really understand the database schema fields. Some of the users, tried searching the schema, but instead of paying attention to the full path of the fields, they simply used the schema to know the name of the field. Some users also forgot to change the image in the start, because they thought that the configuration to show an image would be available inside the settings menu where they write the expression, instead of having to change the type of part first. Even with this low completion rate, the users did not find it very difficult, with a difficulty rating mean of 3.

Tasks 5 and 6 were identical, differing only on what needed to be presented. This made the completion times and rate, very similar between them. These were the last tasks that needed the Expression language, but in a way that is easier than in task 2 and 3, which resulted in a mean difficulty rating of 5. Some users still struggled a little with the task and the expressions but due to it being easy they all completed it. The final task, the number 7, was very easy for the users, as well, with a rating of 5. In this task however, the number of errors was high, because the users did not understand where they needed to put the style to center the column at first, trying in multiple places until it worked.

With the questions asked after the test, the users revealed that they liked the system, and thought that adding blocks and other parts was very easy. In the second question, the users rated the difficulty of the Expression language as a whole, with a mean rating of 4 (mode 5), which means that they did not find the language difficult to understand. The third and fourth questions were related to the difficulty of using variables and fields, which had also a mean rating of 4, but with a lower mode, which reveals the even though they understood the language as a whole, the concepts of variables and fields were more difficult to understand. The last three questions were related with the documentation and revealed that the users thought that it was helpful (this question had the mean and mode of 5, which translated into a strongly agree), understood the fields available in the schema (a 95% number of yesses), and rated the difficulty of understanding the documentation with a mean of 4.

Finally, the System Usability Scale test, puts the application on the top 11.1% of the applications evaluated with this method, using the percentile ranking with the value 68, which according to many studies, is identified as the mean value for the score of a SUS test. For our system the mean rating of the SUS was 81, with a standard deviation of 11. The mean, mode and quartiles were also calculated for these questions, revealing that some users think they need help of a technical person to operate the system. This question had a mean value of 2 (disagree with the necessity of needing help), however, in the third quartile, the response is at a 3, which suggests that 25% of the people may need some help.

In general, some of the comments of the users where related to the icons that allow the edition of the blocks, these icons are small in size and only show when the user moves the mouse over the block to edit. Some said that the icons were too small, or that the way they appear should be changed, for example, making it appear under the cursor on click, instead of moving the whole block to display the controls. All the users said that they felt that with experience it would be easier to perform the tasks asked. Some other users really liked how the data schema viewer in the documentation works, with the live search to find the fields. Some things referred by only one user were that the ”tables were complex” in terms of editing, the auto complete was helpful and that the system had great customization opportunities, but with so many options it can be very complex. One interesting comment was to introduce a button near each property in the configurations, with an example of its usage, instead of having to go to the documentation.

5.3 Discussion

The results in general were good and accomplished the objective of the system of allowing the professors to edit the different aspects of the system, in this case evaluated here, the edit of the views. But even though this was accomplished, there is still room for improvement, especially the problems with the Expression Language, reflected in the Tasks 3 and 4. Some of the most technical users were able to do these tasks, but the ones which did not have experience with programming languages and/or did not understand the concepts explained during the demo had a greater difficulty accomplishing them. This is something that needs to be considered for the future, for example, clarifying the documentation on how the language work could help, or offering examples of the usage near the properties of the views, as suggested by one of the users. One of the problems here was that the users rarely read the documentation in regards to the general use of the Expression Language, because the documentation, which is provided with the system, offers some examples, and if they were to see them it would be easier, because they are very identical to what was asked to do in the tasks.

All the expected times were also below or inside the confidence intervals, which is not a great result, but as explained before, these were specified based on the difficulty of the task (which is somewhat relative) and the pilot test, and in some cases only inside the confidence intervals because of a few seconds, and not above that intervals that would make a great result.

Chapter 6

Conclusion

The motivation for this project was to create a system that can be configurable to decrease those possible negative effects and improve motivation of the students allowing for a better engagement with the gamified course of PCM. This resulted in the objective of our project that was **to create a new dynamic web solution to replace the static pages with an interactive site that allows differentiated forms of presentation to different students** and support the gamified experience of PCM. This objective was accomplished, and using the SmartBoards system, we are now able to create the differentiated forms of presentation for different students, as was our objective. The system allows to create these views in the application using the visual editor which makes it easy for anyone to edit some features of the system. There were a lot of problems and design decisions made during the development that had an impact on the project and the amount of features that the system has. The problems with the views took some time to get sorted, but in the end it turned out great. The expression language used in not very complex in terms of its grammar, and should be easy to learn. In the future, this language could be changed for something more visual, like a visual programming language that does not require as much text.

From the feedback given by the users in the evaluation they say it would be easy to construct a view on their own using the documentation available to aid them. Most of them expressed that they would have performed way better if they had used the system for more time. The analysis of the results were very positive, with a good usability reported by the System Usability Scale test, which had a mean score of 81 and the percentile ranking using an average of 68, found by other studies as being the mean score for the test, puts the system in the top 11.1% of all systems evaluated with this method. From these tests it was also possible to take a lot of advice, from the comments of the users and the way they used the system, to be implemented in the future.

6.1 Future Work

In the future, the implementation of modules to replace the script, would be the next step to turn the SmartBoards into an independent system. Maintaining that link with the script, makes the system somewhat tied to the course of PCM, instead of being able to be used by other courses, because the professor would have to develop all the game elements that are already implemented, but using modules capable processing the information, or modify the script for their use. Another problem that needs to be addressed is the database, which did not end up with the best solution possible. Migrating to a relational database, is probably the best option, but getting access to a server that can host a key-value database management system, would also be good, because the changes would probably be much less.

Some of the suggestions from the users could also be implemented to improve the system. These

suggestions represent what the users want, or what they think can benefit them when using the system, and sometimes are simple to implement. For example, the suggestion of creating a list of the levels unlocked in the Quest, could be implemented with just a few changes to the code. Another example, would be to display the amount of XP since last login in the side bar. This value could be tracked by some module, and made available to the views, through the data schema very easily.

While the main objective of the project was accomplished, the motivation behind it still needs to be tested in the future. Modifying the views, for the students not performing well or which show negative effects with the presence of Gamification, should be approached carefully because it could increase this negative effects, making the students less motivated toward the course. It is up to the professors, or some other future system, to decide what are the markers that indicate the lack of motivation (the types of students discussed in the Related Work could be used, for example) and try to correct them using different views to inspire those people, motivate them and ultimately to succeed.

Bibliography

[1] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "Improving participation and learning with gamification," in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, Gamification '13, (New York, NY, USA), pp. 10–17, ACM, 2013.

[2] 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*, Gamification '13, (New York, NY, USA), pp. 66–73, ACM, 2013.

[3] E. Sun, B. Jones, S. Traca, and M. W. Bos, "Leaderboard position psychology: Counterfactual thinking," in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, (New York, NY, USA), pp. 1217–1222, ACM, 2015.

[4] 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*, MindTrek '11, (New York, NY, USA), pp. 9–15, ACM, 2011.

[5] S. Deterding, "Gamification: Designing for motivation," *interactions*, vol. 19, pp. 14–17, July 2012.

[6] K. Knaving and S. Björk, "Designing for fun and play: Exploring possibilities in design for gamification," in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, Gamification '13, (New York, NY, USA), pp. 131–134, ACM, 2013.

[7] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? – a literature review of empirical studies on gamification," in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pp. 3025–3034, Jan 2014.

[8] M. Meder and B. J. Jain, "The gamification design problem," *CoRR*, vol. abs/1407.0843, 2014.

[9] J. Koivisto and J. Hamari, "Demographic differences in perceived benefits from gamification," *Computers in Human Behavior*, vol. 35, pp. 179 – 188, 2014.

[10] A. Domínguez, J. S. de Navarrete, L. de Marcos, L. Fernández-Sanz, C. Pagés, and J.-J. Martínez-Herráiz, "Gamifying learning experiences: Practical implications and outcomes," *Computers & Education*, vol. 63, pp. 380 – 392, 2013.

[11] R. W. Helms, R. Barneveld, and F. Dalpiaz, "A method for the design of gamified trainings," 2015.

[12] S. O'Donovan, J. Gain, and P. Marais, "A case study in the gamification of a university-level games development course," in *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, SAICSIT '13, (New York, NY, USA), pp. 242–251, ACM, 2013.

[13] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, “Identifying student types in a gamified learning experience,” *International Journal of Game-Based Learning (IJGBL)*, vol. 4, no. 4, pp. 19–36, 2014.

[14] M. Montola, T. Nummenmaa, A. Lucero, M. Boberg, and H. Korhonen, “Applying game achievement systems to enhance user experience in a photo sharing service,” in *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, MindTrek ’09, (New York, NY, USA), pp. 94–97, ACM, 2009.

[15] J. Antin and E. F. Churchill, “Badges in social media: A social psychological perspective,” *CHI 2011 Gamification Workshop Proceedings (Vancouver, BC, Canada, 2011)*, 2011.

[16] C. Butler, “The effect of leaderboard ranking on players’ perception of gaming fun,” in *Online communities and social computing*, pp. 129–136, Springer, 2013.

[17] P. Denny, “The effect of virtual achievements on student engagement,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’13, (New York, NY, USA), pp. 763–772, ACM, 2013.

[18] J. a. P. Costa, R. R. Wehbe, J. Robb, and L. E. Nacke, “Time’s up: Studying leaderboards for engaging punctual behaviour,” in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, Gamification ’13, (New York, NY, USA), pp. 26–33, ACM, 2013.

[19] K. R. Christy and J. Fox, “Leaderboards in a virtual classroom: A test of stereotype threat and social comparison explanations for women’s math performance,” *Computers & Education*, vol. 78, pp. 66 – 77, 2014.

[20] E. D. Mekler, F. Brühlmann, A. N. Tuch, and K. Opwis, “Towards understanding the effects of individual gamification elements on intrinsic motivation and performance,” *Computers in Human Behavior*, pp. –, 2015.

[21] J. Brooke *et al.*, “Sus-a quick and dirty usability scale,” *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

Appendix A

User Tests Data

TASK	AGE	GENDER	COMPLEX	CMS	WEB	Task Time (s) 1	2	3	4	5	6	7
1	24	M	Y	Y	Y	80	100	195		45	20	65
2	25	M	Y	N	N	54	137			88	49	114
3	23	M	Y	Y	N	35	77			78	55	108
4	23	F	Y	Y	Y	46	49	239		48	117	130
5	17	M	N	N	N	67	55	247		104	98	105
6	21	M	Y	Y	Y	39	71	218	160	44	35	68
7	23	M	Y	N	Y	35	62	200		32	73	84
8	23	M	Y	N	Y	85				41	28	160
9	23	M	Y	Y	Y	41	105	245		91	98	83
10	23	M	Y	Y	Y	55	79	253	162	27	78	104
11	22	M	Y	Y	Y	37	54			14	73	80
12	25	M	Y	Y	Y	52	69			44	27	68
13	24	M	Y	N	Y	43	70		146	97	117	84
14	24	M	Y	Y	Y	41	55		89	29	49	79
15	23	M	Y	N	N	83	127	148		59	55	90
16	23	M	Y	N	Y	65	83		189	31	57	134
17	17	M	Y	Y	Y	60	117	177		27	58	53
18	31	M	Y	N	Y	90	90	259	195	27	27	125
19	24	M	Y	N	Y	36	53	119	145	51	27	82
20	20	M	Y	Y	Y	45	47	256	169	43	44	70

Mean:	23				
Std. Dev:	2,9				
Median:	23				
1st Quart:	23				
2nd Quart:	23				
3rd Quart:	24				
Mode:		M	Y	Y	Y
		95%	95%	55%	80%

Mean:	54,5	78,9	213	157	51	59,3	94,3
Std. Dev:	18,2	27,2	46	32,8	26,5	30	27,5
Median:	49	71	229	161	44	55	84
1st Quart:	40,5	55	191	146	30,5	33,3	76,8
2nd Quart:	49	71	229	161	44	55	84
3rd Quart:	65,5	95	249	174	63,8	74,3	110
Xmin(a=0.05):	46	66,2	191	142	38,6	45,2	81,4
Xmax(a=0.05):	62,9	91,7	235	172	63,4	73,3	107

Task Errors							Task Completion							Task Difficulty						
1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
2	0	2	1	1	0	0	Y	Y	Y	N	Y	Y	Y	4	4	4	3	5	5	5
2	1	2	2	0	0	1	Y	Y	N	N	Y	Y	Y	5	4	2	2	3	5	3
0	0	3	1	0	0	2	Y	Y	N	N	Y	Y	Y	5	5	1	1	4	4	2
1	0	2	3	1	1	2	Y	Y	Y	N	Y	Y	Y	5	5	4	2	5	5	3
1	0	1	2	0	0	1	Y	Y	Y	N	Y	Y	Y	5	5	2	1	3	3	3
0	0	1	1	0	0	2	Y	Y	Y	Y	Y	Y	Y	5	4	2	3	5	5	4
0	1	1	2	0	1	1	Y	Y	Y	N	Y	Y	Y	5	5	3	1	5	5	5
1	3	4	2	0	0	2	Y	N	N	N	Y	Y	Y	4	2	3	3	5	5	5
1	2	1	2	1	1	1	Y	Y	Y	N	Y	Y	Y	5	4	4	3	4	4	4
0	0	1	1	0	1	2	Y	Y	Y	Y	Y	Y	Y	5	4	2	3	5	4	3
1	0	1	0	0	1	0	Y	Y	N	N	Y	Y	Y	5	4	1	2	5	3	4
0	0	1	1	1	0	1	Y	Y	N	N	Y	Y	Y	5	5	2	4	2	4	5
1	1	4	1	2	2	1	Y	Y	N	Y	Y	Y	Y	5	5	3	4	5	4	5
0	0	2	0	0	0	0	Y	Y	N	Y	Y	Y	Y	5	5	1	5	5	5	5
2	1	1	1	0	0	0	Y	Y	Y	N	Y	Y	Y	5	4	3	1	4	5	5
1	1	2	2	0	0	2	Y	Y	N	Y	Y	Y	Y	5	5	4	5	5	5	5
1	0	2	1	0	1	1	Y	Y	Y	N	Y	Y	Y	5	4	4	3	5	5	5
1	1	2	1	0	0	1	Y	Y	Y	Y	Y	Y	Y	4	4	3	2	5	5	4
0	0	1	1	1	0	1	Y	Y	Y	Y	Y	Y	Y	5	5	3	4	5	5	5
0	0	2	1	0	0	1	Y	Y	Y	Y	Y	Y	Y	5	5	3	3	5	5	5

Mean:	1	1	2	1	0	0	1
Std. Dev:	1	1	1	1	1	1	1
Median:	1	0	2	1	0	0	1
1st Quart:	0	0	1	1	0	0	1
2nd Quart:	1	0	2	1	0	0	1
3rd Quart:	1	1	2	2	1	1	2

Mode:	Y	Y	Y	N	Y	Y	Y
Completion:	100%	95%	60%	40%	100%	100%	100%

Median:	5	5	3	3	5	5	5
Mode:	5	5	3	3	5	5	5
1st Quart:	5	4	2	2	4	4	4
2nd Quart:	5	5	3	3	5	5	5
3rd Quart:	5	5	3	3	5	5	5

Questions							SUS										SUS Result
1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	9	10	
4	5	5	5	4	Y	4	5	4	5	3	5	1	3	4	5	1	75
5	5	4	3	4	N	2	4	4	2	2	3	1	3	2	2	2	57,5
5	3	3	3	5	Y	5	4	3	3	4	5	1	2	3	4	4	57,5
5	5	4	3	5	Y	5	5	1	5	1	5	1	5	1	4	2	95
5	2	2	2	5	Y	5	5	2	5	2	5	1	5	3	3	3	80
5	3	5	5	5	Y	5	5	2	5	2	5	1	2	2	5	1	85
5	4	4	4	5	Y	4	4	2	3	2	4	1	3	2	4	2	72,5
5	4	4	4	5	Y	4	5	2	5	3	5	2	5	1	4	2	85
5	4	4	3	5	Y	5	3	1	4	1	4	1	3	3	3	4	67,5
5	4	4	4	5	Y	4	5	1	4	3	5	1	4	1	4	2	85
5	3	3	4	3	Y	4	5	1	4	3	5	1	4	1	4	2	85
5	5	4	3	4	Y	5	4	1	4	1	5	1	5	1	4	1	92,5
5	3	3	3	5	Y	5	5	1	5	2	5	1	4	2	5	1	92,5
5	5	4	5	5	Y	5	5	1	4	1	5	1	4	2	4	3	85
5	3	3	4	4	Y	4	5	2	4	3	5	1	5	1	4	2	85
5	5	5	5	5	Y	4	5	1	5	1	5	1	5	1	4	1	97,5
5	4	5	5	4	Y	4	5	1	5	1	4	1	5	1	4	2	92,5
5	4	4	4	5	Y	4	4	1	4	1	4	1	1	1	3	2	75
5	3	2	4	5	Y	2	3	2	4	2	5	1	5	1	3	3	77,5
5	5	4	4	5	Y	4	5	2	3	2	5	1	4	2	4	2	80

Median:	5	4	4	4	5		4
Mode:	5	5	4	4	5	Y	4
1st Quart:	5	3	3	3	4		4
2nd Quart:	5	4	4	4	5		4
3rd Quart:	5	5	4	4	5		5
						95%	

Median:	5	2	4	2	5	1	4	2	4	2
Mode:	5	1	5	2	5	1	5	1	4	2
1st Quart:	4	1	4	1	5	1	3	1	4	2
2nd Quart:	5	2	4	2	5	1	4	2	4	2
3rd Quart:	5	2	5	3	5	1	5	2	4	2

Mean:	81,125
Std. Dev:	11,252924
Median:	85
1st Quart:	75
2nd Quart:	85
3rd Quart:	86,875
Percentile Rank < 68:	11,1%

Appendix B

SmartBoards Documentation

Views

Views allow control over what is displayed in the system. The admins of the course are able to change the registered views in the ways that best suit them.

Types

There are three types of views that can be registered:

- `Single` - simple page that can be configured
- `Role - Single` - page that changes according to the viewer
- `Role - Interaction` - page that changes according to a user and the viewer

Specializations

For the views that rely on a role/user, specializations can be configured in each view tab registered in the Settings menu under the Views general tab. The specializations of roles inherit from other roles according the roles hierarchy, so it is possible to have something in a view that is only shown to a certain user or group of users with a role.

Registering a view

To register a new view all that is required is to get the view handler from the `views` module and use the `registerView` method with the correct parameters.

Example:

```
$viewHandler->registerView($this, 'mycustomview', 'My Custom View', array(
    'type' => ViewHandler::VT_ROLE_INTERACTION
));
```

Views - Parts

Part - Value

This part allows to display a text.

Type - Text

The `Text` type provides a simple text that is static and will always appear the same way.

Type - Field

The `Field` type provides an easy way of accessing fields from the database. It starts by searching for the value and configuring the different keys necessary for the maps. The format allow to format the string for example appending something, or repeating the value. The key for the value in the format is `%v`.

Type - Expression

The `Expression` type provides an advanced way of writing text, it uses the power of the Views Expressions to output highly dynamic text.

Link

When enabled, this allows to define a HTML link for the text. This link can be something complex using an Expression.

Part - Image

This part is similar to the Value part. However it produces an image instead of text.

Part - Block

This part is the main component of any view as it is a container for other parts.

Header

The header, when enabled, displays an image and a text, that serve as the title of the block.

Header - Page Anchor

The Page Anchor defines a non-visible element that is an HTML anchor, that can be used by to jump straight to that part with a link.

Part - Table

This part is a simple table with columns and rows.

Filter

The Filter, when enabled, will allow the user to filter in real time by some variable defined in the `Filter`

input box. This variable must then be defined in every row of the table as a JavaScript variable. It has 2 modes of filtering:

- Hide - which will hide the items that do not pass the filter
- Fade - which will fade the items that do not pass the filter

Sort

The Sort, when enabled, will allow sorting of the columns. There are column by column settings that allow for a finer control of which columns can be sorted, or the default sorting.

Registering a new Part

To implement a new kind of part in the views, it is required to write a Module which will have a JavaScript file which takes care of the runtime of the module and in it must be registered in the PHP in order to process the saving, loading, parsing and processing of the part.

First it is necessary to register the part in the system using the function `registerPartType($partType, $breakFunc, $putTogetherFunc, $parseFunc, $processFunc)` from the `\Modules\Views\ViewHandler` class. This allows the system to recognize the part and process it according to the functions provided. An example of this can be found here.

Next we need to register the part in the front-end of the system, using JavaScript. To do this, we use the `$sbviews` angular service, provided by the views module. An example of the JavaScript file can be found here.

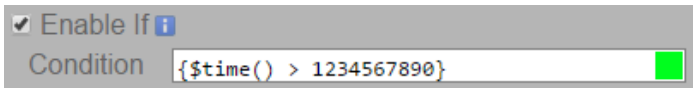
Views - Part Settings

DOM Manipulation - If

The If option is used to choose when to or not to display a certain part. When the `Condition` expression is evaluated as `false` the element will not be visible.

Example

One example of this is to display a Notice. Lets say that the students need to enroll for some activity and you want to alert them. A block with a If option could be created to show the alert message when the enrollment has started.



The expression uses the `$time` function to return the current time. The time is than compared with the number specified as a timestamp, and in the case the condition returns true, the block is displayed.

DOM Manipulation - Repeat

The Repeat part option allows the part to be repeated using some container (array or map) as the provider of the items. The `Elements` field must contain an expression that returns one such container. And the `Key` field must contain a simple string (only letters A through Z), that which will define a few Expression Variables. The variables are:

- `<key>key` - original key of the array or map;
- `<key>pos` - sorted position of the item in array or map;
- `<key>` - Continuation of the value of the item.

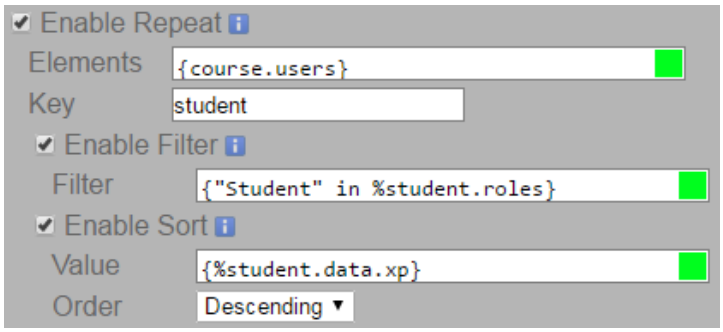
When enabled, the Filter option allows to filter items from the container, repeating only for the items for which the expression defined in the `Filter` field is evaluated as `true`.

The Sort option allows to sort the items of the container, according to the resulting value of the Expression in the `Value` field. It is possible to change the direction of the sort, using the `Order` select box.

Note: The repeat position variable can not be used in the expressions of the `Filter` field, nor in the sort `Value` field.

Example

For example, to configure a list like the leaderboard, we would do this:



The Elements expression will repeat for all users in the course and assign them to the variable student. This Elements are filtered using the Filter expression to contain only the users that have "Student" in their roles, and sorted using the XP data value.

Styling/Misc - Style

Using the `Style` expression field it is possible to change the way parts look in the view. It is simple to add a CSS rule to manipulate the part as we wish.

Styling/Misc - Class

The `Class` expression field is similar to the Style, however it requieres the user to write a Module and setup the CSS file containing the CSS classes.

Styling/Misc - Angular Directive

The `Angular Directive` expression field is used to specify an AngularJS directive to be used in the part. This is a necessary part for the use of Events. However, it is required to write a Module and setup a JS file to use this feature.

Styling/Misc - Events

The `Events` option allows to control the interaction of the user with the Parts. To add an event select an event from the list, and press `Add`. In the box next to the event name, write an Angular Expression to be evaluated in the scope of the part. To have a custom function to be called in the scope, it is necessary to define an Angular Directive.

Styling/Misc - Data

The `Data` option allows to define variables that are can be used for other features of the views. After adding a new key, it is possible to define an expression in the box next to the key name. This expression will be evaluated and the value will be saved in the variable.

Using the selection box that is present after the Expression box, it is possible to select the context of the variable. If the context is `JavaScript`, the variable will be stored in the Data of the element in the DOM, which can be easily accessed from a Angular Directive or a function that is defined for an Event. In the case of the context `Variable` the variable is used by the PHP expression evaluator, and can be used to define variables to use in the part or its children, if the part allows.

Views - Expressions

First Steps

To configure the SmartBoards views in an advanced way, it is required to learn how to use Expressions, which are very similar to a programming language.

An example of such expression is `Hello {users[%viewer].name}!` which would be evaluated as 'Hello John!', for example, if the name of the viewer was John.

Let's break down how it works. Anything you type is accepted as text, except if it starts with a `%` (percentage) which will get the value of a variable. To escape this case use `%%`. When you want to evaluate something it should be placed inside curly brackets, for example, `{2*3}`. To access the name of the viewer as we demonstrated in the first example, we use `{users[%viewer].name}` which basically goes to the **map of users** gets the one with the key **%viewer** (which is evaluated as the id of the viewer, predefined by the view) and them returns the **name** of that user. The data schema can be viewed in the Schema documentation page.

Operations/Functions

As in many other languages the expressions provide the simple mathematics operators for addition `+`,

subtraction `+`, multiplication `*`, division `/` and modulo `%`. It also provides logic and bitwise operators, and `&&` (bitwise `&`), or `||` (bitwise `|`), negation `!` (bitwise `~`) and bitwise xor `^`.

It is also possible to call functions defined in PHP from the expressions. A function call starts with `$` and is followed by an arbitrary number of letters, A through Z. The arguments of the function are enclosed in parentesis `()` and need to be comma separated.

These functions need to be registered by a module, however there are some predefined functions:

- $value(continuation) - Returns the value of the continuation
- $size(array/map) - Returns the size of the map or array
- $urlify(string) - Removes Spaces of a String
- $time() - Returns a timestamp
- $formatDate(string time) - Returns the time formated
- $if(cond, valueOne, valueTwo) - Executes a if condition
- $isModuleEnabled(module) - Returns true if the module is enabled
- $getModules() - Returns an array continuation with all enabled modules

Examples (one per line):

```
$time()
$urlify("Challenger of the Unknown")
$formatDate($time())
```

Variables

Variables are used by the expressions to allow an easier creation of expressions without having to write very long expressions to access certain fields for example. As discussed in First Steps, a variable starts with a `%` (percentage symbol) and is followed by an arbitrary number of letters, A through Z.

The value of a variable can be a single value, for example, a number or a string, but it can also be an object that is returned by a function. There is a special kind of object called a **Continuation** which allow the variable to be used as a path, for example, `{%skill.name}`, if the skill was a continuation of some other path, provided by a DOM Manipulation - Repeat of the view. (For module developers: It is possible to create continuations from arrays with arbitrary keys, for more information, consult Returning a Continuation)

Data Access

To access the data saved in the SmartBoards system, its just a matter of knowing the path for the data. The data schema can be consulted here. To access the data in the path all that is needed is to specify that path inside a expression like `{users[12345].name}`, which in this case would get the name of the user with id 12345.

Developers can see how to define accessible data in the data schema here.

Developing a function

Custom functions can be used in the expressions, whoever these must be defined using PHP and a module (if you need to setup the module take a look at the First Steps on how to setup a module). After setting up a module on the initialization we need request the module `views`, get the view handler and register the function.

```
public function init() {
    $viewHandler = $this->getParent()->getModule('views')->getViewHandler();
    $viewHandler->registerFunction('concat', function($param1, $param2) {
         return new Modules\Views\Expression\ValueNode($param1 . $param2);
    });
}
```

These functions should return a ValueNode that can be presented on the page, or return a Continuation to be used by other parts and functions.

Returning a Continuation

A continuation can be returned on the function, from an arbitrary PHP array, by using the provided function `\Smartboards\DataRetrieverContinuation::buildForArray($array)` that will build the necessary structures and return the DataRetrieverContinuation object.

Charts

Part - Chart

This Chart part allows to display different types of charts to the users. There are a 4 types of charts:

- Line - draws a line chart
- Bar - draws a bar chart
- Star - draws a star plot
- Progress - draws a progress bar

To create a new chart it is required to register a chart in using a Module in PHP.

Register a new chart

To register a new chart, get the `chart` Module, and then invoke the `registerChart($id, $processFunction)` method, which receives the new chart id and the process function that will return the values of the chart.

```
$chartsModule = $this->getParent()->getModule('charts');
$chartsModule->registerChart('xpWorld', function(&$chart, $params, $visitor) {
    //..
});
```

Modules

Module Creation

Creating a module is necessary if you want to use advanced features on the SmartBoards, like defining functions or directives for Views, or to extend functionality of the SmartBoards in a modular way.

Creating an empty module

All you need to get started is the following code in a file with the name `module.EmptyModule.php` inside a new directory in the directory of the modules. It is recommended that this directory uses a lowercase name, or it might not work in some systems.

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

class EmptyModule extends Module {
}

ModuleLoader::registerModule(array(
    'id' => 'empty',
    'name' => 'My Empty Module',
    'version' => '0.1',
    'factory' => function() {
        return new EmptyModule();
    }
));
```

This code, creates an empty PHP class that is the module and registers it with the module loader, which will load it whenever it is required.

On Initialization

To initialize something when the module initializes the function `init()` is used. It allows the module to use other features of the SmartBoards core, like Navigation, or setup additional things required for the module.

As an example, lets define a new navigation button, which is only available for Administrators of the system. To do this, we start by defining our empty `init()` function inside the class.

```
class EmptyModule extends Module {
    public function init() {
    }
}
```

Then we state that we need to use something that is part of the Core of the SmartBoards system by appending `use SmartBoards\Core` near the other use statements. An to conclude it we add the code that checks if the user is an Administrator and in that case add the navigation, resulting in the final code:

```
<?php
use SmartBoards\Core;
use SmartBoards\Module;
use SmartBoards\ModuleLoader;

class CustomNavigation extends Module {
    public function init() {
        $user = Core::getLoggedUser();
        if (($user != null && $user->isAdmin()))
                                Core::addNavigation('images/gear.svg',   'Técnico',
'http://tecnico.ulisboa.pt');
    }
}

ModuleLoader::registerModule(array(
    'id' => 'customnav',
    'name' => 'Custom Navigation',
    'version' => '0.1',
    'factory' => function() {
        return new CustomNavigation();
    }
));
```

The example can be downloaded here, ready to use, just needs to be extracted to the modules folder.

Setup Resources

If you want to define a plugin that applies CSS or needs Javascript, you need to specify which resources you want to include. This is done using the function `setupResources()` and using the function `addResources()` from the Module class to add the resources, which can be entire folders.

```
public function setupResources() {
    parent::addResources('css/'); // some styles
    parent::addResources('js/main.js'); // add main.js first
    parent::addResources('js/'); // other files depend on the main
}
```

Other classes from the Module

To use other classes in the module, you can include them manually, or use the class loader provided by the SmartBoards system. This class loader will load the file if the namespace starts with the name `Modules` followed by the directory of the plugin, for example, `namespace Modules/Views;`.

The classes can then be created normally and are auto loaded by the system automatically, but considerations on the dependencies of the modules should be taken into account, due to the load order of the classes.

Interacting with other modules

To interact with other modules we simply request the module we want by using the function `getModule($module)` from the course. The requested module should be using the class loader way of loading classes, as explained in Other classes from the Module and provide some kind of API for other modules to interact with it.

As an example we can request the module `views` and get the view handler.

```
public function init() {
    $viewHandler = $this->getParent()->getModule('views')->getViewHandler();
}
```

Defining accessible data in the Data Schema

To define some type in the Data Schema some functions of the class `\SmartBoards\DataSchema` are used. There are 4 different data types that can be registered. The types and functions used to create the objects to register in the data schema are in the following list:

- Value/Field - `DataSchema::makeField`
- Array - `DataSchema::makeArray`
- Object - `DataSchema::makeObject`
- Map - `DataSchema::makeMap`

The functions `DataSchema::courseModuleDataFields` and `DataSchema::courseUserDataFields` exist to help create fields for the module and user.

After obtaining the objects it is required to register those objects in the schema, using the function `DataSchema::register`.

```
public function init() {
    DataSchema::register(array(
        DataSchema::courseModuleDataFields($this, array(
            DataSchema::makeArray('levels', null,
                DataSchema::makeObject('level', null, array(
                    DataSchema::makeField('minxp', 'Min XP', 2000),
                        DataSchema::makeField('title', 'Title of the Level', 'Self-
Aware')
                ))
            )
        ))
    ));
}
```

Accessing Module Data

To access the module data, all that is needed to do is invoke the `getData` function from module class.

```
$this->getData()->set('test', true');
```

Accessing User Data

To access user data it is necessary to get the user and then invoke the `CourseUser::getData` function to access the data.

```
$this->getParent()->getUser(12345)->getData()->set('test', true');
```

In the example we get the user from the parent of the module, which is the course. Then we get the data from the user, which is wrapped by a DataWrapper, that manages the save automatically.

API

The API allows to get and update any information that is stored by the SmartBoards system.

To access the API it is required to have an API Key. There are 2 different types of keys for accessing the data. The first type is bound to the global system, which authenticates queries and updates to global settings. The second type is bound to courses and allows to change course information. Each course key is different.

API Endpoint: <SmartboardsBase>/api/

Authorization

There are different ways of sending the authorization key to the server:

HTTP Header

The key can be sent in the authorization header of the HTTP, like in so many other APIs.

Example on request to `http://smartboards/api/`:

```
Authorization: Bearer 1e0ad42e1c66d2324278575ab2c70a46
```

GET

The key can be sent in a GET parameter of the HTTP request.

Example:

```
http://smartboards/api/?key=1e0ad42e1c66d2324278575ab2c70a46
```

POST

THe key can be sent in the body of the POST, in a json object with the key.

Example on POST request to `http://smartboards/api/`:

```
{"key": "1e0ad42e1c66d2324278575ab2c70a46"}
```

The following errors can be generated if something is wrong:

- `MissingAuth` - if no key was provided
- `InvalidKey` - if the provided key is invalid

Request

The Request API function to be called is specified in the GET parameter of the HTTP Request with the key `request`.

Example:

```
http://smartboards/api/?request=users
```

Query (GET)

A query request allows to get information from the system. It is done using a GET HTTP Request, authorized in the Authorization section, and with the request parameter. Additional paremeters may be required.

Update (POST)

An update request allows to update information in the system. It is done using a POST HTTP Request, authorized in the Authorization section, and with the request parameter. Additional paremeters may be required.

The body of the POST request must be a JSON object.

The following errors can be generated if something is wrong:

- `MissingRequest` - if no request key is specified
- `UnknownRequest` - if the request does not match any registered function

API Functions

Global Functions

The Global functions require the Global API Key for authorization.

The following functions are available:

- `config` - general config of the SmartBoards system
- `users` - all users

All functions in this section implement Query and Update requests. To update from what is returned use the key `update` in the JSON payload of the POST request.

Example Update: `{"update": {"headerLink":"Hello"}}`

Course Functions

The Course functions require the Course API Key for authorization and a `course` parameter in the GET or POST to specify the course.

The following functions are available:

- `course` - get course info and users
- `course-user-data` - get course user data - requires a `user` parameter
- `course-module-data` - get course module data - requires a `module` parameter

All functions in this section implement Query and Update requests. To update from what is returned use the key `update` in the JSON payload of the POST request.

Example Update: `{"update": {"headerLink":"Hello"}}`