Computer Science and Information Systems 13(3):809–826 DOI: 10.2298/CSIS160123025P

Gamification of Learning Activities with the Odin service

José Carlos Paiva[1], José Paulo Leal[2], and Ricardo Queirós[3]

[1] CRACS & INESC-Porto LA, Faculty of Sciences,
University of Porto, Porto, Portugal up201200272@alunos.dcc.fc.up.pt
[2] CRACS & INESC-Porto LA, Faculty of Sciences,
University of Porto, Porto, Portugal zp@dcc.fc.up.pt
[3] CRACS & INESC-Porto LA & DI/ESEIG/IPP,
Porto, Portugal ricardoqueiros@eseig.ipp.pt

Abstract. Existing gamification services have features that preclude their use by e-learning tools. Odin is a gamification service that mimics the API of state-of-the-art services without these limitations. This paper presents Odin as a gamification service for learning activities, describes its role in an e-learning system architecture requiring gamification, and details its implementation. The validation of Odin involved the creation of a small e-learning game, integrated in a Learning Management System (LMS) using the Learning Tools Interoperability (LTI) specification. Odin was also integrated in an e-learning tool that provides formative assessment in online and hybrid courses in an adaptive and engaging way.

Keywords: Gamification, E-Learning, Game services, Interoperability.

1. Introduction

The use of game concepts and mechanics in non-game contexts is an effective way to engage users. Gamification is currently a word of order in different domains, from marketing to e-learning [2]. The massive use of this approach led to the concept of gamification as a service, provided by major players such as Google[4] and Microsoft[5]. These services leverage on their large user base to provide support for game progress mechanics such as points, leaderboards and badges, without requiring a specific authentication from the client application.

Gamification services are a great advantage to small web and tablet based applications, in particular to games. The game progress mechanics features provided by these services are also relevant in e-learning. However, e-learning systems are typically deployed in environments with a single sign-on managed by an academic institution. For practical but also ethical reasons, it would be unacceptable to require students to have an account with a third party such as Google, for instance.

In this paper we present Odin as a gamification service for learning activities. This is an extended version of a paper [6] first presented at the Symposium on Languages, Applications and Technologies – SLATE 2015. Odin is a gamification service similar to the state of the art, without requiring registration of the end users. Its API is inspired in the Google Play Game Service (GPGS) with minor adjustments regarding user identification.

[4] https://developers.google.com/games/services
[5] http://azure.microsoft.com/en-us/documentation/services/mobile-services/

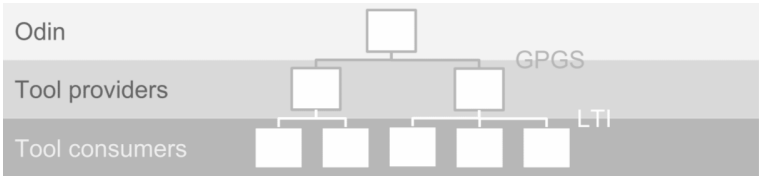

Fig. 1. Architecture of e-learning systems using Odin

Odin exposes its services to educational games used in an ecosystem of e-learning systems based on a Learning Management System (LMS), and was designed to have a pivotal role in these systems, as depicted in Figure 1. The systems implementing these games can be seen as tool providers for another layer of e-learning systems, typically Learning Management Systems (LMS) that provide user authentication. The communication between these layers uses different APIs: tool providers interact with Odin using a variant of the GPGS API, and consumers interact with tool providers using the Learning Tool Interoperability (LTI) API [11].

The remainder of this paper is organized as follows. Section 2 reviews the state of the art in game services. Section 3 introduces the Odin service, its design and implementation. Section 4 describes its evaluation using a small serious game as case study. Finally, Section 5 summarizes the contributions of this research and identifies opportunities for future work in this project.

2. Game Services

The video game industry is one of the fastest growing sectors in the worldwide economy [14]. According to the research company Gartner, global video game sales may have reached $111.1 billion in 2015, due in part to the growth in mobile game play and the recent release of the new generation of game consoles. In order to increase engagement and player retention, video games include several common features such as leaderboards and achievements. The massive use of this approach and the impressive growth of the number of players led to the concept of gamification as a service, later materialized in Game Backend as a Service (GBaaS). The approach is simple. Instead of replicating the implementation of the game features in each version of the game for various platforms, GBaaS adhere to a service oriented architecture providing cross-platform game services that enable the integration of popular gaming features such as achievements, leaderboards, remote storage and real-time multiplayer in mobile games.

While the concept of ”winners and losers” can hinder the motivation of students [13], gamification is currently being applied with relative success in e-learning [1, 12]. The integration of game concepts in learning environments helps students to remain focused and to fulfill their course goals. However, the implementation of gamification in these domains is often trapped in ad-hoc solutions or supported by specific platforms (for instance, the badges in Moodle), instead of using approaches such as those provided by GBaaS.

In the following subsections, we briefly summarize the main common game features that can be applied to the teaching-learning process. Then, we compare eight GBaaS regarding social and technical features. This study is part of an effort to select a GBaaS

implementation on which to base the development of a service for gamification of learning activities.

2.1. Game concepts

Games are more interesting when players are able to achieve goals and compete against other players. These features foster retention and competitiveness, and are applicable also in the gamification of e-learning activities. The following list enumerates the most common game concepts:

Leaderboards are databases that keep scores. They allow users to post their scores in a game and compare themselves with other players' scores. They measure the success of a player in a game.

Achievements are goals/challenges set in a game that players managed to accomplish. Achievements give players a motivation to keep playing, to earn as many points as possible, and a are way to compare themselves with other players. The fulfilment of a goal may enhance the status of the player or unlock access to other levels, for instance.

Multiplayer is a play mode that allows several players to simultaneously cooperate or compete in a game. This feature supports a range of other sub-features, such as challenges, where players compete with each other on either a score challenge or an achievement challenge, and matchmaking games in real-time, turn-based, or self-hosted matches.

Saved games allow the remote storage (in the cloud) of game data, for instance, the state and the players' progress in the game.

Quests are periodic game challenges that players can complete to earn rewards. This way, developers can launch periodic challenges to their gaming communities.

Gifts allow players to send/request game resources or items to/from friends (for instance, in their Google+ circles).

Matchmaking automatically sets up game matches and finds opponents based on parameters set by the game developer. Usually only a specific number of players can be matched at the same time.

2.2. Game Backend Services

A **Backend-as-a-Service (BaaS)** is a cloud computing service model acting as a middleware component that allows developers to connect their Web and mobile applications to cloud services via application programming interfaces (API) and software development kits (SDK). BaaS features include cloud storage, push notifications, server code, user and file management, social networking integration, location services, and user management as well as many other backend services. These services have their own API, allowing them to be integrated into applications in fairly simple way [3].

A **Game-Backend-as-a-Service (GBaaS)** is a subset of a BaaS that includes cross-platform solutions for the typical game concepts identified in the previous subsection. During the development process of a game (or a generic application) developers must choose between building their own backend services or using an available game backend platform. This last option is usually preferred since GBaaS include several services

specifically tailored for game development. These services allow developers to focus on the game logic by freeing them from implementing boilerplate features.

The following subsections compare several GBaaS implementations according to their social and technical features. Given the number of GBaaS implementations found (32) it would be impracticable to study them all. Therefore, eight GBaaS implementations were chosen: Google Play Game Services, Yahoo Backend Game Service, GameUp, Flox, GameSparks, Fresvii, Kumakore and Photon. These features are summarized in Table 1 and Table 2.

Social game features The studied GBaaS implementations provide developers with social game services through a cross-platform API. These features make the gameplay more competitive and collaborative, and improve social engagement.

Table 1. Social game features

	Google	Yahoo	GameUp	Flox	GameSparks	Fresvii	Kumakore	Photon
Leaderboards	yes	no	yes	yes	yes	yes	yes	yes
Achievements	yes	yes	yes	no	yes	no	yes	no
Multiplayer	yes	yes	no	no	yes	yes	yes	yes
Save Data	yes	yes	yes	no	yes	yes	yes	yes
Quests	yes	no	no	no	yes	no	no	yes
Gifts	yes	yes	no	no	yes	no	yes	yes
Matchmaking	no	no	yes	no	no	yes	no	yes

Analysing Table 1 one concludes that almost all GBaaS implementations support leaderboards, multiplayer game mode and cloud storage. Other features such as quests and matchmaking are not yet widely supported, probably due to their novelty.

Technical game features The studied GBaaS offer cloud services through API and SDK to various platforms. Table 2 compares the eight GBaaS regarding the authentication method, web service flavours, resource format and platforms supported.

Regarding authentication almost all GBaaS implementations use the same strategy. Before making any calls to the game services, the game must first establish an asynchronous connection with the backend servers and authenticate itself within the game services. Some GBaaS require the players to have an account on specific backends (GPGS requires users to have a Google account). Others, such as GameSparks, provide a simple mechanism that allows games to implement social login without any additional code, allowing gamers, for instance, to sign in using a Facebook or Twitter account, and start playing.

The majority of the GBaaS implementations provides an HTTP RESTful API. The format of the data in all HTTP store operations (PUT and POST) is required to be valid JSON. All response data from the GBaaS comes back also in JSON format. In fact, JSON is becoming the data exchange format of choice [10] due to its simplicity and terseness, particularly when compared with XML. Regarding the REST API reference, the authors' opinion is that GPGS is the most complete and best documented API.

Table 2. Technical game features

	Users authentication	Web Service	Resource format	Platforms
Google	Google+	REST	JSON	Android / iOS / C++
Yahoo	Yahoo/Facebook	-	-	ActionScript / iOS / Android / C# / Unity
GameUp	Facebook	REST	JSON	Android / iOS / FirefoxOS / Unity
Flox	Google+ GameCenter Facebook	REST	JSON	ActionScript / iOS / JavaScript
GameSparks	Facebook Twitter	REST	JSON	Unity / Marmalade / Cocos2D / JavaScript / ActionScript / C++
Freshvii	Facebook	-	-	Android / iOS / Unity
Kumakore	Facebook	REST	JSON	Android / iOS / Unity / .NET
Photon	Facebook	REST	JSON	Android / .NET / Unity

In addition to the REST API, most GBaaS implementations support also mobiles. There are examples of SDKs for Android, iOS, and even FirefoxOS (GameUp) mobile native apps. Game engines are also supported and most GBaaS implementations offer SDKs for major game engines such as Unity, and also for cross-platform game development tools such as Marmalade and Cocos2D.

3. Odin

This section describes Odin, a gamification RESTful Web Service intended to be used by educational institutions. It provides (1) score submissions, (2) leaderboards listing, (3) quests for players, (4) awards to players for in-game accomplishments as well as some minor services to manage institutions, players, leaderboards, quests and achievements.

Odin is based on a standard gamification API but has a different approach regarding authentication. Institutions, rather than end-users, are the ones that require authentication. Once an institution is authenticated, Odin grants it permission to manage scores, quests and achievements on behalf of its users.

The next subsections present the architecture of Odin and its main components, and describe its data model and service API.

3.1. Architecture

Odin is a RESTful Web Service that allows institutions to consume gamification resources from their web applications. The web applications initialize sessions in Odin through authentication built on top of OAuth2 authorization protocol[6]. Then they request particular actions to the server identified by a specific URI and an HTTP method such as POST, GET, PUT or DELETE.

[6] http://oauth.net/2/

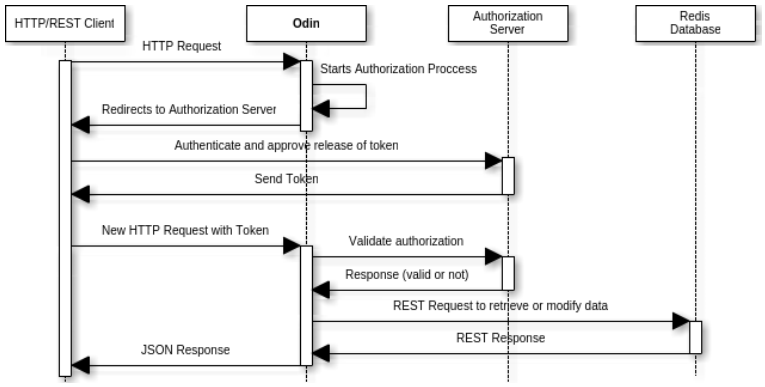

Fig. 2. Sequence diagram representing a common request to Odin

Figure 2 presents a sequence diagram that summarizes the interactions of Odin with other systems when a request is made by the client. Firstly, the HTTP request made by the client is subject to a security filter that checks if the institution is authenticated. If the institution is not authenticated or authorized to access Odin resources, it is redirected to the authorization server where it will authenticate and approve the release of an authorization token. The generated token (with expiration time) is sent to the client that presents it to Odin.

When the client is authenticated and authorized, it is passed to the JAX-RS REST interface implemented using Jersey (described in the next subsection) and forwarded to the mapped resource. From the resource layer it is forwarded to the service layer, passing through a security layer which intercepts it to check authorization and roles, ensuring that only authorized institutions have access to the services.

The service layer responds to the request with the data persisted on Redis (described in the next subsection) through the Jedis client (using REdis Serialization Protocol) and a Ohm library implementation for Java. The response sent to the client is a JSON object representing the resource type modified or requested by it (each resource type may have one or more data representations, as detailed in subsection 3.4). Whenever a fresh token is needed, the client can request it from the Authorization Server.

3.2. Frameworks and Tools

Odin uses Jersey[7], an open-source framework that is the reference implementation of the Java API for RESTful Web Services, extending it with additional features and utilities to further simplify RESTful service. Among other features, Jersey provides a *Core Server* to build RESTful services based on annotations, support for JSON and to the Java Architecture for XML Binding, as well as a *Core Client* to easily create a client that can communicate with REST services.

Data storage relies upon the Redis NoSQL database[8] that provides an open-source and advanced key-value storage and cache solution. It is an high performance alternative to the

[7] https://jersey.java.net/
[8] http://redis.io/

traditional Relational Database Management Systems (RDBMS) [9] to store and access a large amount of data. Redis is sometimes described as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. As a NoSQL database it focus on performance and scalability rather than in guaranteeing the atomicity, consistency, isolation and durability (ACID) properties. Redis was selected for backend due to its ability to store large amounts of non critical data very efficiently.

In order to integrate Redis in Odin the data layer resorts to the Jedis client[9], as well as of an object-hash mapping library, named JOhm[10], to store and retrieve objects from Redis with an higher level of abstraction and thus simplicity. JOhm is the Java implementation of the well-known Ohm library[11] and aims to be minimally-invasive, relying only on hooks for persistence. These hooks are specified by Java annotations and implemented using reflection.

3.3. Data Model

The data model of Odin consists of seven main entities: institution, player, leaderboard, score, quest, achievement and session. The relationships among these entities are shown by the UML class diagram of Figure 3.

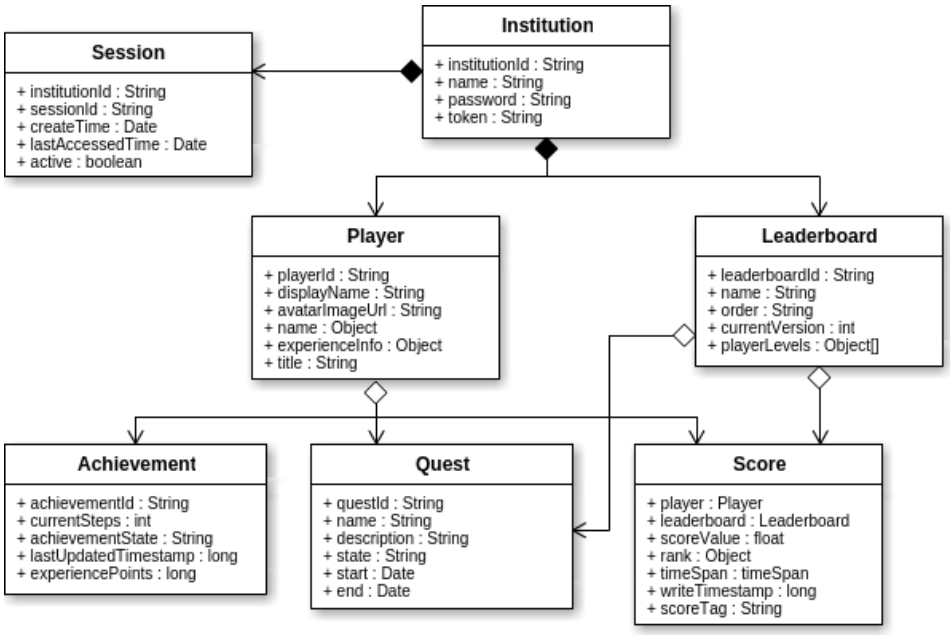

Fig. 3. Class diagram of the data model of Odin

[9] https://github.com/xetorthio/jedis
[10] https://github.com/agrison/johm a fork from https://github.com/xetorthio/johm
[11] http://ohm.keyvalue.org/

An institution is the entity that manages games and all related data, and so it is the one which needs authentication and/or authorization. Thus, it needs to store an id and password to authenticate, and also a token to check the validity of the session. Whenever an institution authenticates a session is created and linked to it (through the *institutionId*). This session contains the creation time, last access time and a state (active or inactive).

The institution needs to represent its students. As this is a gamification model they are abstracted to players, and so they will have a *playerId* that identifies them to the institution, a *displayName* that is the name to show on the leaderboard, a full name and a representation of his experience info with level, points acquired and points needed to move up to the next level.

As the player progresses in the game, (s)he will eventually earn achievements. An achievement has a number of required steps and a state (hidden, revealed or unlocked). When a player reveals one, he receives the associated number of experience points.

A player can also accept and fulfil quests. A quest is characterized by a name, a description, a state (upcoming, open, accepted, completed, failed, expired or deleted), a start and an end date.

One of the most important parts of this model is the leaderboard. It contains more than a list of sorted scores, it contains data related to a game, such as a list of info on the levels available in the game/leaderboard. These entities are connected since a single leaderboard is required to each game, and they depend on the existence of each other.

Scores related to a leaderboard and a player, are also stored. Each score has a floating point value, a timespan (daily, weekly or all time score), a timestamp and a rank (its position on the leaderboard).

3.4. Service API

The integration of Odin with other systems relies on REST calls to set and retrieve data. It follows the Google Play Games Services API Reference[12] for achievements, leaderboards, players, quests and scores resources. The only differences are that all these resources' URI paths are relative to gamify/institutions/**institutionId**. Also when an authenticated player is referenced in a method, it is replaced by a sub-path of the form /players/**playerId** right after **institutionId** in the resource path URI.

The institution resource is added to the set of resources. It contains the functions shown in table 3.

Table 3. Institutions resource API reference. URIs are relative to /gamify

Function	HTTP request
insert	POST /institutions
get	GET /institutions/institutionId

The `insert` function inserts the institution given in the request body. The `get` function retrieves the institution resource given its id. Tables 4 to 8 describe part of the reference API of Odin, organized by resource type (each resource type can have many data representations and methods).

[12] https://developers.google.com/games/services/web/api/index

Table 4. Score resource API reference. URIs are relative to /gamify/institutions/institutionId/

Function	HTTP request
get	GET /players/playerId/leaderboards/leaderboardId/scores/timeSpan
list	GET /leaderboards/leaderboardId/scores
	Required query parameters: timeSpan
listWindow	GET /leaderboards/leaderboardId/window
	Required query parameters: timeSpan, playerId
submit	POST /leaderboards/leaderboardId/scores
	Required query parameters: score
submitMultiple	POST /leaderboards/scores

Table 4 presents the reference API of the score resource. The `get` function retrieves high scores, and optionally ranks, for the given player (for a specific timespan the path parameter timespan must be set to the required timespan, otherwise it must be set to ALL). The `list` function lists the scores in the leaderboard for a given timespan. The `listWindow` function lists the scores in a leaderboard near to the given player's score. The `submit` function submits a score to the specified leaderboard. The `submitMultiple` function submits multiple scores to leaderboards.

Table 5. Leaderboard resource API reference. URIs are relative to /gamify/institution/institutionId

Function	HTTP request
get	GET /leaderboards/leaderboardId
insert	POST /leaderboards
list	GET /leaderboards

The reference API of the leaderboard resource is presented in Table 5. The `get` function retrieves the metadata of the leaderboard with the given id. The `insert` function inserts the leaderboard passed as POST parameter. The `list` function lists all leaderboards of the institution.

Table 6. Players resource API reference. URIs are relative to /gamify/institutions/institutionId

Function	HTTP request
insert	POST /players
get	GET /players/playerId
list	GET /players

Table 6 summarises the reference API of the player resource. The `insert` function inserts the player given in the request body. The `get` function retrieves the player resource with the given id. The `list` function retrieves all the players of the given institution.

Table 7 introduces a few functions of the achievement resource. The `increment` function increments the steps of the achievement for the requested player and leaderboard. All achievements' progress of a player can be listed with the `list` function. The `reveal` func-

Table 7. Achievements resource API reference. URIs are relative to /gamify/institution/institutionId/players/playerId

Function	HTTP request
increment	POST /achievements/achievementId/increment
	Required query parameters: stepsToIncrement, leaderboardId
list	GET /achievements
	Required query parameters: leaderboardId
reveal	POST /achievements/achievementId/reveal
	Required query parameters: leaderboardId
setStepsAtLeast	POST /achievements/achievementId/setStepsAtLeast
	Required query parameters: steps, leaderboardId
unlock	POST /achievements/achievementId/unlock
	Required query parameters: leaderboardId
updateMultiple	POST /achievements/updateMultiple
	Required query parameters: leaderboardId

tion reveals an achievement, given its id and the id of the player. The `setStepsAtLeast` function sets the steps of the given player towards unlocking an achievement (if the steps parameter is less than the already stored steps, the achievement remains unchanged). The `unlock` function unlocks the achievement with the given id in the given player. It is also possible to update multiple achievements at once through the `updateMultiple` function that receives an object with a list of achievements to be updated as a value of the key `updatedAchievements`.

Table 8. Quests resource API reference. URIs are relative to /gamify/institutions/institutionId/leaderboards/leaderboardId

Function	HTTP request
accept	POST /players/playerId/quests/questId/accept
list	GET /players/playerId/quests

Finally, Table 8 presents part of the reference API of the quest resource. The `accept` function indicates that the player with given id will participate in the quest. The `list` function enumerates the quests for a given player and leaderboard.

The data exchange format of Odin is JSON. All data passed in HTTP store operations (PUT and POST) is required to be valid JSON. Also, all data in the response body of any HTTP request is in JSON format.

Each resource type can have different data representations, depending on the method and/or direction of the message (request or response). As an example, the scores resource has a distinct data representation in the request body of the method to submit multiple scores (presented in Figure 4), when compared to the data representation in the method to list all scores in a leaderboard (presented in Figure 5).

```
{
    "kind": "gamify#scoreSubmission",
    "leaderboardId": string,
    "playerId": string,
    "score": float,
    "scoreTag": string
}
```

Fig. 4. Data representation of a score resource - request body of Submit Multiple Scores

```
{
    "kind": "gamify#leaderboardEntry",
    "player": players Resource,
    "scoreRank": long,
    "formattedScoreRank": string,
    "scoreValue": float,
    "formattedScore": string,
    "timeSpan": string,
    "writeTimestampMillis": long,
    "scoreTag": string
}
```

Fig. 5. Data representation of a score resource - response body of List Scores

A Reference API documentation[13] of Odin is already available, although not yet complete. It contains use case examples, for each of its functions, for two types of consumers, Jersey Client and `curl` command via Unix shell.

4. Evaluation

The evaluation of Odin was twofold. First, we created a simple game to validate the concept. Then, we have integrated it in an e-learning tool for learning programming languages. This integration served as an evaluation of the user experience. In the next subsections we describe these two validations.

4.1. Proof of Concept

For validation of the concept of the gamification service described in the previous section, a simple multiplication game was created. This game – MathGamify – can be used by primary school children to learn multiplication tables. MathGamify generates two random numbers. The first number between 1 and the current game level and the second number between 1 and 10. Then the student/player has the opportunity to answer the multiplication value of the two numbers. The score is accumulated in the ratio of the player's level until player misses, in which case the score is reset to zero.

[13] http://odin.dcc.fc.up.pt:8080/odin-web-api

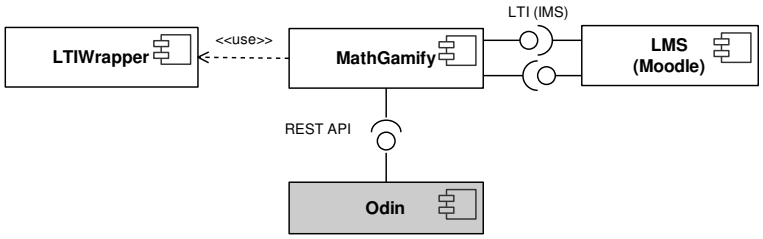

Fig. 6. MathGamify component diagram

Figure 6 presents the component diagram of MathGamify. MathGamify acts as a tool provider to a Learning Management System (LMS). The integration of MathGamify with the LMS relies on the Learning Tools Interoperability (LTI) specification. When the LMS launches MathGamify the LTI parameters are sent as part of the HTTP POST request. On request reception MathGamify uses the LTI Wrapper [7] package to process LTI communication and extract user id, name and level. The last is a custom parameter defined on the external tool configuration of the LMS.

MathGamify uses two functions of Odin: score submission and listing of scores. Once the player answers a question, MathGamify communicates the score to Odin, using Jersey Client to issue the REST call, and the grade to the LMS using LTI. This grade is a value between 0 and 1, calculated as follows: if there is a custom parameter `custom_max_score` then it is the score divided by `custom_max_score`, otherwise it is the number of correct answers divided by the total number of tries. When MathGamify initializes its GUI, and every time a score is submitted, the score listing is updated with the data returned from Odin.

One of the key components to the implementation of MathGamify is the LTI Wrapper [8], a Java package developed for integrating another tool with an LMS but that can be used by any Java application requiring LTI communication. It implements both sides of the LTI communication, by receiving LTI requests from LMS and issuing LTI requests to LMS.

The GUI component of MathGamify was developed using Google Web Toolkit (GWT), an open source Java software development framework that allows a fast development of AJAX applications in Java. The GWT code is organized in two main packages, the server and the client. The server package includes all the service implementations triggered by the user interface. These implementations are responsible of (1) the logic of the game, (2) communication with Odin and (3) communication with LMS through LTI wrapper. The selected LMS was Moodle 2.8 [14].

The implementation of MathGamify demonstrates the efficacy of the proposed approach in coping with the extra requirements of a serious game integrated in a typical e-learning ecosystem, where authentication is provided by an LMS. To complement its validation, Odin was also tested regarding its efficiency.

The latency of the Odin service was tested in three of its functions: (1) insert a player at an institution, (2) submit a single score and (3) list an ordered page of scores of the

[14] https://moodle.org/

leaderboard with either, 25 records and 1000 records. Each test consisted of 1000 samples of calls to the same function. In test (1) and (2) random usernames and a random score value were generated per sample.

Initially the tests were run locally on the same machine as the Odin server, using Grizzly Test Container provided by Jersey, so they had no network latency. The same tests were repeated on an external server, running Apache Tomcat 8.0.24. During these tests an average network latency of 26.47 ms and a maximum of 112.2 was observed. The time statistics, in milliseconds, for each test and for each test container are presented in Table 9.

Table 9. Results of efficiency test to Odin

Test Container	Function/Task	Maximum (ms)	Average (ms)	Median (ms)
Grizzly (local)	Insert player	274	3.79	3
	Add score	77	3.44	3
	List scores (1000 scores' page)	814	412.09	415
	List scores (25 scores' page)	532	182.78	184
External	Insert player	349	54.15	52
	Add score	1057	55.62	53
	List scores (1000 scores' page)	1324	758.29	721
	List scores (25 scores' page)	1434	381.1	383

The tool used to measure time spent was ContiPerf[15], a lightweight testing utility that allows the user to easily turn JUnit 4 test cases into performance tests. It is based on annotations as the JUnit 4's test configuration.

4.2. User Experience Evaluation

The main goal of Odin was to be integrated in an e-learning system. This goal was already accomplished, since Odin was integrated in Enki (not yet described in the literature), a web-based IDE for learning programming languages. This IDE blends assessment and learning, presenting content, from hypertext to video, as well as exercises, in an adaptive and engaging way.

Enki uses gamification to engage students in the learning process. It interacts with Odin to support the creation of leaderboards, reward students for their achievements, among others. Besides its integration with Odin, Enki also integrates with several other services and tools, namely, a service for sequencing educational resources to provide different learning paces according to students' capabilities, a learning management system, an evaluator engine which marks and grades exercises, an exercise creator to ease the creation of exercises and a learning objects repository to store educational resources. These last three tools are already part of Mooshak 2.0 (the new version of Mooshak [4]), the system which hosts Enki.

[15] http://databene.org/contiperf

Figure 7 presents the diagram of the network of Enki, where Odin plays an important role, by providing the gamification features to this e-learning system, through a REST API.

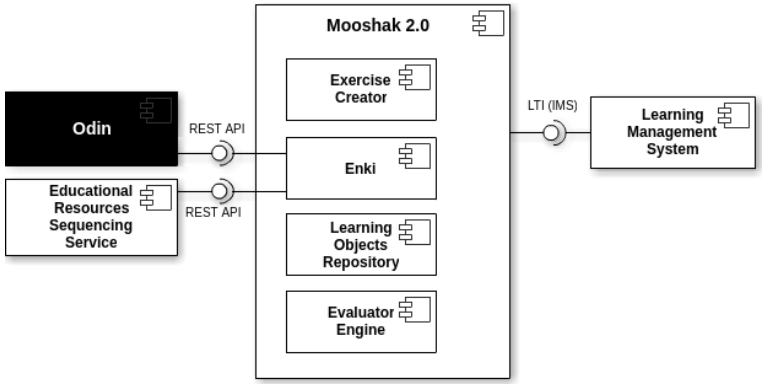

Fig. 7. Network of Enki with Odin highlighted

Odin is also an important part of the Enki's interface (which we present in Figure 8 with focus on Odin). The top-right region windows are mostly built based on Odin's data. This region aggregates several windows to engage students, such as a leaderboard, a list of achievements and statistics of a problem.

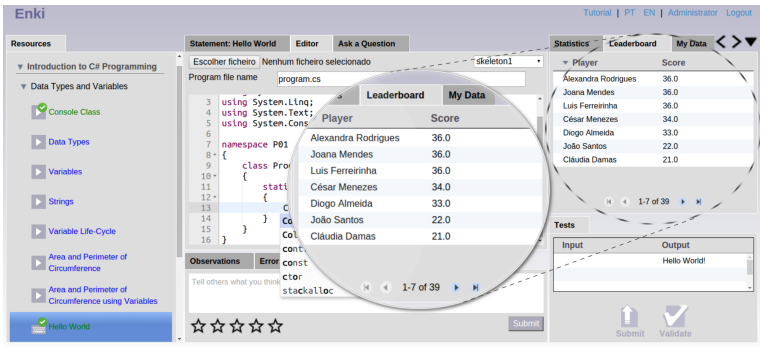

Fig. 8. Interface of Enki with focus on Odin

The global system of Enki was subject to an acceptability evaluation, which also served as a user experience evaluation of Odin. This evaluation consisted of an experiment with undergraduate students of *Escola Superior de Estudos Industriais e de Gesto (ESEIG)* - a school of the Polytechnic Institute of Porto.

The experiment was carried out from the 4th to the 15th of January of 2016 and took the form of an Open Online Course under the subject of "Introduction to C# Program-

ming". It was free of charge and without registration limits. A total of 70 students were enrolled in the course, of which 28 were females. The course contains expository and evaluative resources.

The expository resources are mainly videos presenting solutions to example problems and some course topics. These videos were created with a software which records screen activity and voice – Camtasia[16] –, and deployed on YouTube. These videos had some requirements upon their creation. They should cover all the curricula (coverage), have several difficulty levels (diversity), have at most 5 minutes (fragmentation), and be composed of pictures, sound, and subtitles (completeness).

The evaluative resources are programming exercises that allow students to apply and test their knowledge. They comply with the Mooshak programming problem package specification. This package is an archive containing a problem description (typically an HTML file), a file with the solution, an XML file with the structure of the package, a folder with tests and their output, and optionally a folder with images and a folder with skeletons of the solution.

The majority of the students that, effectively, entered in Enki has tried to solve all the evaluative resources. They have also submitted many questions to the authors. These facts show that Odin fosters competitiveness between the students and/or the course is able to engage students.

After the experiment, the students were invited to fill-in an online questionnaire, using Google Forms, based on the Nielsen's model [5]. This survey includes questions on the utility and usability of the complete system of Enki. Nielsen has defined the utility as the capacity of the system to achieve the desired goal and the usability as a qualitative attribute that estimates how easy is to use an user interface. A total of 25 students has completed the questionnaire, of which 9 were females.

Figure 9 summarizes the results of the survey in a bar chart, grouped by Nielsen's heuristics and then, sorted in descending order of user satisfaction. These results showed that the heuristics with higher user satisfaction were the consistence, recognition and aesthetic. However, they also highlighted problems in three areas: speed and reliability, error prevention and users help and documentation.

The last question of the survey consists of an overall classification of Enki. There are 5 options from which the student should choose one: very good, good, adequate, bad, very bad. Most of the students (56%) classified Enki as an adequate tool and a great part of them (40%) stated Enki as a good or a very good tool. Only a few students (4%) considered it either bad or very bad.

Although students complained about some deficiencies in this system, such as the delay to validate or submit their programs and the lack of documentation, none of these negative sides are related to Odin itself. Furthermore, many of the students classified Enki as a good or a very good tool largely due to its gamification features.

5. Conclusions

Game concepts and mechanics are an useful way to engage students in e-learning activities. These kind of features are already provided by game backend services that leverage

[16] https://www.techsmith.com/camtasia.html

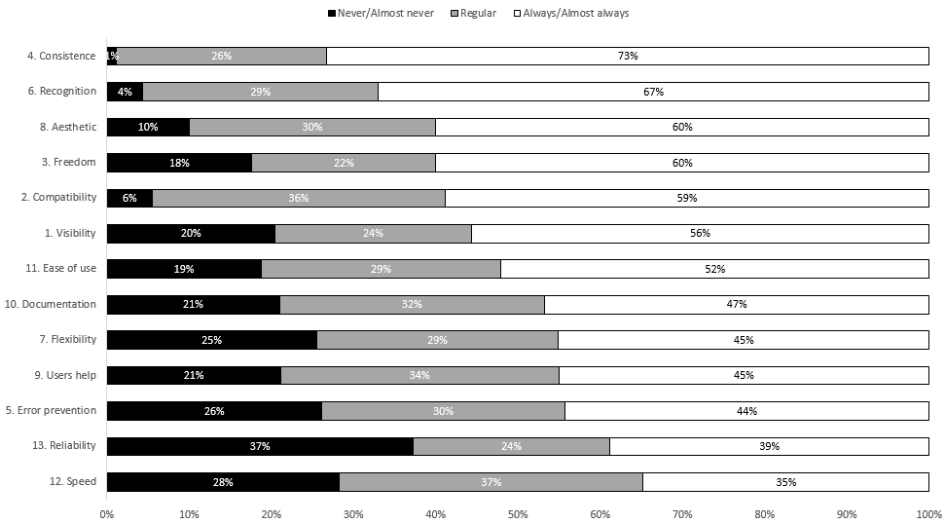

Fig. 9. Summary of the acceptability of Enki

on their authentication services and massive user base. However, gamification services that rely on external authentication are not adequate for e-learning systems that already operate on a single sign-on ecosystem.

Odin is a gamification service developed for the requirements of e-learning systems. It was designed to authenticate clients rather than end-users and thus can be integrated with the e-learning systems typically found in educational institutions.

The MathGamify system is a proof of concept, that illustrates how educational games acting as tool providers for an LMS interact with the services of Odin. The authors have also integrated Odin in an actual e-learning tool. This tool, called Enki, interfaces with Odin to support the creation of leaderboards, reward students for their achievements, among others.

Enki is an e-learning tool for formative assessment in online and hybrid courses. It is a web IDE that blends assessment and learning, presenting content, from hypertext to video, as well as exercises in an adaptive and engaging way. It uses gamification to engage students in the learning process. It also integrates with a service for sequencing educational resources to provide different learning rhythms according to students' needs. The exercises and assessment of Enki are, typically, computer programs.

Odin itself will be subject to improvements. The current version provides web services for exposing the gamification service to clients. The next version will provide also a web interface to register institutions and allow them to manage their resources.

Acknowledgments. This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.

References

1. Burguillo, J.C.: Using game theory and competition-based learning to stimulate student motivation and performance. Comput. Educ. 55(2), 566–575 (Sep 2010), http://dx.doi.org/10.1016/j.compedu.2010.02.018
2. Hamari, J., Koivisto, J., Sarsa, H.: Does gamification work?–a literature review of empirical studies on gamification. In: System Sciences (HICSS), 2014 47th Hawaii International Conference on. pp. 3025–3034. IEEE (2014)
3. Janssen, C.: Backend-as-a-service (baas)". Tech. rep., Techopedia, http://www.techopedia.com/definition/29428/backend-as-a-service-baas (2014)
4. Leal, J.P., Silva, F.: Mooshak: a web-based multi-site programming contest system. Software: Practice and Experience 33(6), 567–581 (2003), http://dx.doi.org/10.1002/spe.522
5. Nielsen, J., Landauer, T.K.: A mathematical model of the finding of usability problems. In: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems. pp. 206–213. CHI '93, ACM, New York, NY, USA (1993), http://doi.acm.org/10.1145/169059.169166
6. Paiva, J.C., Leal, J.P., Queirós, R.: Odin: A service for gamification of learning activities. In: Languages, Applications and Technologies, pp. 194–204. Springer (2015)
7. Queirós, R., Leal, J.P., Campos, J.: Sequencing educational resources with seqins. Computer Science and Information Systems 11(4), 1479–1497 (2014)
8. Queirós, R., Leal, J.P., Paiva, J.C.: Integrating rich learning applications in lms. In: State-of-the-Art and Future Directions of Smart Learning, pp. 381–386. Springer (2016)
9. Seeger, M., Ultra-Large-Sites, S.: Key-value stores: a practical overview. Computer Science and Media, Stuttgart (2009)
10. Severance, C.: Discovering javascript object notation. Computer 45(4), 6–8 (2012)
11. Severance, C., Hanss, T., Hardin, J.: Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools. Technology, Instruction, Cognition and Learning 7(3-4), 245–262 (2010)
12. Siddiqui, A., Khan, M., Akhtar, S.: Supply chain simulator: A scenario-based educational tool to enhance student learning. Comput. Educ. 51(1), 252–261 (Aug 2008), http://dx.doi.org/10.1016/j.compedu.2007.05.008
13. Vansteenkiste, M., Deci, E.L.: Competitively contingent rewards and intrinsic motivation: Can losers remain motivated? Motivation and Emotion 27, 273–299 (2003), http://dx.doi.org/10.1023/A:1026259005264, 10.1023/A:1026259005264
14. Zackariasson, P., Wilson, T.: The Video Game Industry: Formation, Present State, and Future. Taylor & Francis (2012), http://books.google.pt/books?id=lgiQNdc-DOwC

José Carlos Paiva is a student at the department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP) and holder of a research grant of the Center for Research in Advanced Computing Systems (CRACS). He has interest in the development of eLearning systems and gamification.

José Paulo Leal is an assistant professor at the department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP) and associate researcher of the Center for Research in Advanced Computing Systems (CRACS). His main research interests are eLearning system implementation, semantic web and software engineering. He has a special interest on automatic exercise evaluation, in particular on the evaluation of programming exercises, and on web adaptability. He has participated in several research

projects in his main research areas, including technology transfer projects with industrial partners.

Ricardo Queirós is an assistant professor at the Department of Informatics of the School of Management and Industrial Studies (Polytechnic Institute of Porto). He is also a researcher in the field of e-learning standardization and interoperability, XML Languages and e-learning systems architectural integration at the Center for Research in Advanced Computing Systems (CRACS) research group of INESC TEC Porto and at the Knowledge Management, Interactive and Learning Technologies (KMILT) research group. He has participated in several research projects in his main research areas, including technology transfer projects with industrial partners.

Received: January 23, 2016; Accepted: July 31, 2016.