UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Thiago de Campos

UMA PLATAFORMA PARA INTEGRAÇÃO E PROCESSAMENTO DE REGRAS E ELEMENTOS DE GAMIFICAÇÃO EM SISTEMAS

Florianópolis - SC
2018/1

THIAGO DE CAMPOS

UMA PLATAFORMA PARA INTEGRAÇÃO E VALIDAÇÃO DE REGRAS E ELEMENTOS DE GAMIFICAÇÃO EM SISTEMAS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção de grau de Bacharel de Sistemas de informação

Prof. José Eduardo de Lucca

Florianópolis - SC
2018/1

Thiago de campos

UMA PLATAFORMA PARA INTEGRAÇÃO E PROCESSAMENTO DE REGRAS E ELEMENTOS DE GAMIFICAÇÃO EM SISTEMAS

Trabalho de conclusão de curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharelado em Sistemas de Informação.

Orientador:

_____

Prof. José Eduardo De Lucca
UFSC

Banca Examinadora:

_____

Prof. Dr. Leandro José Komosinski
UFSC

_____

Profa. Frank Augusto Siqueira
UFSC

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MDA - Mechanics, Dynamics and Aesthetics

RPG - Role Playing game

NPC's - Non-player character

API - Application programming interface

REST - Representational State Transfer

XP - Experience points

GPS - Global Positioning System

MIT - Massachusetts Institute of Technology

HTTP - HyperText Transfer Protocol

JPA - Java Persistence Aplication

CSS - Cascading Style Sheets

HTML - HyperText Markup Language

RESUMO

Gamificação é uma prática que tem por objetivo engajar e motivar pessoas em tarefas e contextos do dia-a-dia através de características e elementos de jogos. Empresas estão usando esta técnica para atrair e manter clientes, melhorar o desempenho de funcionários e colaboradores, obter mais sucesso em treinamentos e métodos de ensino, entre outros. Todavia, a elaboração de uma estratégia de gamificação eficiente e duradoura, que desperte a motivação intrínseca das pessoas, requer uma série de análise e estudos que acabam sendo o grande desafio para quem deseja obter resultados positivos. A proposta deste trabalho é criar uma ferramenta de fácil integração, que facilite o desenvolvimento de ambientes gamificados. Para isso, foi desenvolvida uma plataforma gerenciável, unindo a criação de elementos de jogos, como pontos de experiência, níveis, insígnias, atributos, com a criação e processamento de regras que visam automatizar a atribuição destes elementos aos usuários do sistema. Com um painel administrativo e um sistema de *feedback* simples, a solução desenvolvida pode servir como uma alternativa a outras plataformas do mercado.

Palavras-chave: gamificação, plataforma de gamificação, elementos de jogos, jogos, engine, motivação, engajamento.

ABSTRACT

Gamification is a practice that aims to engage and motivate people in tasks and day-to-day contexts through game features and elements. Companies are using this technique to attract and retain customers, improve employee performance, gain more success in training and teaching methods, and more. However, the development of an efficient and long-lasting gamification strategy, which arouses people's intrinsic motivation, requires a series of analyzes and studies that end up being the great challenge for those who want to obtain positive results. The purpose of this work is to create an easy integration tool that facilitates the development of gamic environments. For this, a manageable platform was developed that combines the creation of game elements, such as experience points, levels, badges, attributes, with the creation and processing of rules that aim to automate the assignment of these elements to system users. With an administrative dashboard and the a simple feedback system, the developed solution can serve as an alternative to other market platforms.

Keywords: gamification, gamification platform , game elements, games, engine, motivation, engagement.

1. INTRODUÇÃO

Com a invenção do computador e posteriormente o surgimento dos jogos eletrônicos, as possibilidades acerca da criação e desenvolvimento de jogos aumentaram drasticamente, dando a esse mercado um crescimento e relevância que perduram até hoje, como mostram alguns levantamentos apresentados por Bohyun Kim, em *Understanding Gamification (2015)*. Do computador aos consoles, e posteriormente celulares, *smartphones*, *tablets* e dispositivos de realidade virtual, os jogos estão presentes em praticamente todas as camadas da sociedade (KIRRIEMUIR e MCFARLANE, 2004; WILLIAMSON, 2009), sendo hoje, como aponta o estudo feito pela PwC em relação a 2016, a maior indústria no setor de entretenimento nos Estados Unidos.

O grande aumento da aquisição de smartphones e computadores pessoais nos últimos anos, junto com o crescente número de pessoas com acesso à internet móvel, vem fazendo com que as pessoas estejam cada vez mais conectadas e acostumadas a interagirem através deste meio (KIM, 2015). Unindo esta nova maneira de viver e interagir da sociedade com a capacidade dos jogos de proporcionar fortes experiências e engajar pessoas, novas abordagens em torno deste mercado acabaram surgindo, como é o caso de gamificação (do inglês, *gamification*), um conceito que apesar de já ser usado há muito tempo em outros moldes, só começou a ser explorado e tomar a forma que se conhece hoje em meados de 2002 (VIANNA *et al.* 2013).

A técnica de gamificação pode ser entendida de diferentes formas por diferentes pessoas, entretanto, apesar de possuir diversas similaridades com os

jogos, gamificação não se trata de criar ou jogar um jogo. De acordo com Zichermann e Cunningham (2011), McGonigal (2012), Kim (2015) e diversos outros autores da literatura - e que também é o conceito adotado neste trabalho -, gamificação é uma técnica com o objetivo de extrair os elementos positivos dos jogos, tais como pontuação, conquistas, objetivos, regras, mecânicas, dinâmicas, sistema de *feedback*, etc., e inseri-los em outros tipos de atividades e contextos no intuito de torná-los mais atrativos, e por consequência, provocar uma mudança de comportamento nas pessoas envolvidas, tornando-as mais engajadas, motivadas e satisfeitas.

Seguindo essas premissas, por exemplo, uma empresa poderia “gamificar” sua ferramenta de gerenciamento interno a fim de tornar o processo como um todo mais envolvente e promover um fator social entre os funcionários da empresa de modo que se sintam mais motivados a usar corretamente a ferramenta. Da mesma forma, atividades que precisam ser feitas pontualmente e que normalmente encontram relutância das pessoas que necessitam praticá-las, tais como exercícios físicos e rotinas de tratamento de doenças (principalmente em crianças), poderiam fazer uso de gamificação em aplicativos auxiliares a fim de tornarem-se mais prazerosas e divertidas, como proposto por André Matias em seu trabalho de conclusão do curso de Sistemas de Informação da UFSC (2015), onde desenvolveu um aplicativo para auxiliar crianças no controle de diabetes. Além do setor empresarial e da área de saúde e bem estar, a técnica de gamificação também é encontrada em diversos outros segmentos da sociedade, como educação, meio ambiente, marketing, finanças, agendas, vendas, etc.

Apesar do conceito ser simples, a incorporação de gamificação em um determinado contexto não é uma tarefa trivial. Para alcançar os resultados desejados, ou seja, envolver as pessoas em uma experiência lúdica e fazer com que engajem na atividade proposta, é preciso, além de traçar uma boa estratégia de execução, planejar com muito cuidado os diversos aspectos que compõem uma boa estratégia de gamificação (KIETZMANN et al., 2016). De acordo Zichermann e Cunningham (2011), Vianna (2013) e outros autores da literatura, construir abordagem de gamificação de sucesso envolve estudar e conhecer os tipos de jogadores[1] e de que forma podem impactar na experiência; ser capaz de escolher as mecânicas de jogos e saber quais estão alinhadas com os objetivos do projeto, além do interesse particular que os diferentes tipos de jogadores nutrem por cada uma delas; pensar sobre que tipo de emoções se deseja despertar nas pessoas e entender como as dinâmicas e aspectos visuais da gamificação podem ajudar nesta questão; estudar como, quando e quanto recompensar os jogadores pelas suas ações e quais ações serão recompensadas; elaborar as mensagens de *feedback* e quando serão mostradas, além de outros aspectos que podem vir a ser importantes dependendo da complexidade do projeto, como a narrativa e o balanceamento (KIETZMANN, 2016).

Todos esses fatores críticos da construção de uma experiência gamificada culminam em um resultado que tem por objetivo motivar e engajar pessoas. Entretanto, o fator motivacional pode ter diversas origens, podendo resultar em dois tipos de motivação: extrínseca e intrínseca (KIM, 2015). De acordo com Edward L.

[1] Vianna, Zichermann e outros autores da literatura classificam as pessoas que participam do processo de gamificação em diferentes tipos de jogadores de acordo com as categorias de mecânicas que mais às atraem em um jogo: competitividade, colaboração, exploração e socialização..

(*et al*. 1999, apud KIM, 2015) a motivação é extrínseca quando uma pessoa realiza uma ação buscando algo externo à ação, como uma recompensa (dinheiro, status, poder, etc.). Ao contrário, a motivação intrínseca é guiada pela vontade de realizar a própria ação, sem depender de elementos externos ou qualquer tipo de recompensa extra. Nesse caso, a própria ação se torna a recompensa, pois tem um significado especial para o indivíduo (KIM, 2015). Para gamificação esse é um assunto importante, pois a grande maioria das abordagens costuma seguir o caminho da motivação extrínseca, ou seja, oferecendo recompensas a cada ação importante dos usuários. Acontece que na prática a motivação extrínseca é fraca, pois se sustenta em elementos externos que ao serem removidos causam um desinteresse imediato do usuário (ZICHERMANN e CUNNINGHAM, 2011). Ainda assim, não significa que as recompensas não possam ser usadas em uma experiência de gamificação, mas sim, que elas não devem ser o principal atrativo. É preciso achar um meio termo.

Diante de tantos fatores a serem considerados e planejados, é natural que em muitos casos, a dificuldade em se implementar uma boa estratégia de gamificação em um sistema se encontra mais na concepção da ideia do que no desenvolvimento do software propriamente dito. A finalidade deste trabalho é justamente amenizar a parte técnica, propondo como solução uma plataforma que atuará como um facilitador e ferramenta de auxílio para a implantação de gamificação em sistemas, tirando parte do peso da implementação de sistemas gamificados, servindo também como um guia de gamificação através das funcionalidades disponíveis e do fluxo com que o sistema é configurado.

No decorrer deste trabalho, serão abordados os diversos aspectos que definem o conceito de gamificação, começando pela evolução dos jogos e o surgimento de gamificação, mostrando como esta técnica é utilizada nos dias atuais, abordando aspectos humanos como engajamento, motivação e como isso está relacionado com as diferentes tipologias de jogador identificadas na literatura. Em seguida, os elementos de jogos são apresentados, dando enfoque nos principais dentro do conceito de gamificação, para que então, exemplos de outras soluções existentes nesta área sejam analisados. O trabalho encerra com a especificação e o desenvolvimento da plataforma, onde é apresentado cada um dos componentes e conceitos criados, bem como a interação entre cada um destes componentes, finalizando com os possíveis trabalhos futuros e considerações finais.

1.1 JUSTIFICATIVA

Como abordado na introdução, criar ou adaptar um sistema dentro do conceito de gamificação envolve não só a compreensão dos mecanismos de gamificação e suas relações com o engajamento e motivação dos jogadores, mas também o desenvolvimento do software que dará vida a esses elementos. Como visto também, apesar de sempre ser importante ter um software de qualidade, o sucesso ou não da gamificação está mais atrelado à concepção e bom planejamento da ideia do que das tecnologias que serão utilizadas para pô-la em prática (ZICHERMANN e CUNNINGHAM, 2011). Todavia, isso não significa que implementar um sistema gamificado seja uma tarefa trivial. Existem diversas mecânicas de jogos que podem ser usadas e cada uma delas possui suas próprias especificações e regras que devem ser respeitadas. Além disso, para manter os

dados dos usuários e elementos da gamificação organizados, atualizados e otimizados, é preciso ter uma boa estrutura de persistência criada especialmente para resolver o problema.

É válido tomar nota também que qualquer abordagem de gamificação, ainda que seja extremamente simples ou que utilize diversos outros recursos e técnicas conhecidas em gamificação, acaba sempre usando algumas mecânicas de jogos já conhecidas, que apesar de não ser a única coisa importante dentro de uma abordagem, é de fato um ingrediente presente em todas elas. Cada uma destas mecânicas necessita de um tipo de tratamento e estrutura de validação diferente.

Com base nestas colocações, é natural pensar em usar soluções prontas que tratam desses aspectos como APIs, *frameworks*, bibliotecas de software, *engines*, etc. Este trabalho propõe uma plataforma que procura explorar justamente essa necessidade de facilitar a introdução dos elementos jogos em sistemas e softwares existentes ou em construção, bem como qualquer tipo de regra ou condições de uso atrelado a esses elementos. Além disso, a plataforma pode servir como inspiração e ponto de partida para a concepção da ideia de gamificação em si, visto que muitos conceitos importantes estão enraizados no projeto. A ideia é que com isso, mais projetos que tenham a intenção de usar gamificação, mas que não dispõem de tempo, recursos ou conhecimento a respeito do assunto, possam ser viabilizados ou pelo menos facilitados.

1.2 OBJETIVOS

1.2.1 Objetivo geral

- Desenvolver uma plataforma para facilitar a integração de mecanismos de gamificação em aplicações, sistemas, páginas web e softwares em geral.

1.2.2 Objetivos específicos

- Disponibilizar os serviços atrelados à plataforma através de uma uma API que servirá como via de comunicação entre o cliente e a plataforma.
- Desenvolver uma estrutura de banco de dados apropriada, onde serão guardados os dados dos usuários (jogadores), as regras de gamificação e os elementos de jogos do sistema.
- Prover um guia de uso da plataforma, bem como a documentação completa da API.

2. REFERENCIAL TEÓRICO

2.1 A EVOLUÇÃO DOS JOGOS E SUA INFLUÊNCIA NA SOCIEDADE

De acordo com historiador Huzinga (1950) (*apud* KOLB; KOLB, 2010), o ato de jogar vem acompanhando o ser humano desde muito antes dos primeiros manuscritos e está presente em praticamente todas as nossas áreas de empreendimento, tais como a ciência, o direito, a filosofia, a guerra e a arte. Os gregos com os jogos olímpicos da antiguidade e os romanos com o duelo de gladiadores são apenas alguns dos exemplos mais evidentes (VIANNA *et al.* 2013).

Com a invenção do computador, os jogos também acabaram ocupando seu espaço no meio digital, dando seus primeiros passos em meados de 1960 quando Steve Russell criou o primeiro jogo digital chamado *spacewar*, na universidade do MTI (RABIN, 2010). Após esse período, o desenvolvimento de jogos digitais e produtos relacionados nunca parou de crescer, se expandindo para os mais variados meios, desde os *arcades* e consoles, até os celulares e computadores pessoais (RABIN, 2010).

Hoje, o mercado de jogos é um dos que mais cresce no mundo [1]. Um levantamento feita pela Newzoo, empresa especializada em pesquisa de mercado de entretenimento digital e *mobile*, indica que para o ano de 2016 o mercado de jogos moveu aproximadamente US$ 99,6 bilhões, um total de 8,4% a mais em relação à 2015 [2]. Nos Estados Unidos, outra pesquisa feita pela PwC, indica que a partir de 2016 o crescimento anual da indústria de jogos irá superar o de outras áreas de entretenimento como cinema, música, e literatura até 2020 [3].

De acordo com Bohyun Kim (2015), hoje, outros três fatores contribuem para o crescimento contínuo da indústria de jogos e para o surgimento de novas abordagens: (1) a rápida adoção do *smartphone* pela sociedade, (2) o enorme crescimento da web em mobiles, e (3) o crescimento das redes sociais. Uma pesquisa feita pela *Paw Research Center* [19] para o ano de 2014 aponta que nos Estados Unidos 83% dos jovens entre 18-29 anos e 74% dos adultos entre 30-49 anos possuem um *smartphone* (KIM, 2015).

Somando a aptidão dos seres humanos aos jogos e os três fatores mencionados - o smartphone, a *web mobile*, e as redes sociais -, é natural imaginar que os jogos também estejam inseridos neste meio, ainda mais se levarmos em conta o poder de processamento e armazenamento dos *smartphones* atuais em relação aos celulares mais antigos e a introdução de novos recursos como o GPS, por exemplo (KIM, 2015). Porém, além de aparecem em sua forma clássica, ou seja, com o objetivo puro de entreter, os jogos também passaram a ser aproveitados para a criação de novas abordagens e conceitos de mercado, como é o caso da gamificação, uma técnica que aplica os elementos positivos dos jogos para aumentar o engajamento das pessoas em atividades ou processos do cotidiano (VIANNA *et al.* 2013).

2.2 GAMIFICAÇÃO

O termo gamificação (do original em inglês *gamification*) foi mencionado pela primeira vez por Nick Pelling em 2002 (SZYMA, 2013 apud KIM, 2015), mas só passou a ser realmente conhecido e ter sua adoção generalizada em 2010 (Deterding *et al.* 2011 apud KIM, 2015).

Apesar de *gamification* ser um termo originado da palavra *game* (jogo), o que propõe uma certa relação entre os dois conceitos, é seguro dizer que apesar desta relação existir, jogo e gamificação não são a mesma coisa. Na prática, o processo de gamificação não consiste em fazer um jogo, mas sim, apoderar-se de suas mecânicas, dinâmicas e demais elementos positivos e inseri-los em outros ambientes, atividades ou processos no intuito de motivar, engajar, emocionar e mudar o comportamento das pessoas envolvidas (VIANNA 2013; ZICHERMANN e CUNNINGHAM, 2011; KIM, 2015).

Para que as pessoas de fato mudem seu comportamento em relação a alguma atividade ou sintam-se atraídas pelos elementos de jogos inseridos em um ambiente que normalmente não é visto como jogo, a técnica de gamificação pressupõe, assim como nos jogos, o uso de dois elementos importantes: o sistema de recompensas e o sistema de *feedback* (ZICHERMANN e CUNNINGHAM, 2011; KIM, 2015). O modo como estes dois elementos aparecem e são usados em um processo de gamificação geralmente segue um fluxo de acontecimentos baseado em ação e consequência da seguinte forma: (1) Alguma ação é realizada pelo usuário ou público alvo. (2) Se a ação realizada é importante e deve ser estimulada, uma recompensa é fornecida. (3) Ao ser recompensada, a pessoa recebe um *feedback* instantaneamente, para que saiba que a ação que acabou de executar é importante ou desejada (ZICHERMANN e CUNNINGHAM, 2011).

O Foursquare, um aplicativo lançado em 2009, foi um dos principais precursores no uso de gamificação. O Foursquare aplicou a técnica a uma rede social onde o objetivo era o compartilhamento de informações sobre locais (ZICHERMANN e CUNNINGHAM, 2011; KIM, 2015). Na vida real, fora do mundo

dos jogos, as pessoas estão o tempo todo visitando lugares, seja para comer, trabalhar, fazer compras, estudar, se divertir, etc. O Foursquare combinou a técnica gamificação e o uso do GPS dos *smartphones* para tornar essa prática em algo mais social. O processo é simples: Ao visitar um lugar, os usuários usam o aplicativo para marcar que estiveram naquele local, fazendo um *checkin*, e em seguida podem deixar dicas e comentários sobre aquele lugar, como: “a batata frita é ruim, mas a cerveja é barata”. Essa informação é compartilhada em redes sociais e dentro do círculo de amizades no próprio aplicativo. A cada *checkin* em um lugar, o aplicativo imediatamente dá um feedback ao usuário e, dependendo dos lugares pode atribuir recompensas simbólicas, como medalhas e conquistas, e até mesmo recompensas físicas, como um café grátis. Quanto mais *checkin*s o usuário fizer em um local, maior será sua “reputação”, podendo alcançar o título de “*major*”, um rótulo concedido às pessoas com números de *checkins* muito alto em um local [22].

Em síntese, usando o conceito de gamificação descrito anteriormente, podemos analisar o uso da técnica no Foursquare da seguinte forma (KIM, 2015): mecânicas de jogos - medalhas, conquistas e pontuação -, são aplicadas à um contexto do mundo real - a prática de visitar lugares -, no intuito de motivar as pessoas a usarem o aplicativo para pesquisar lugares, e interagirem com outros usuários. Para isso, o aplicativo conta com um sistema de recompensas e um sistema de *feedback* atrelados à ação de fazer *checkin*. Apesar de não ser um jogo, os elementos de gamificação presentes no Foursquare foram capazes de tornar a rede social em uma espécie de competição e socialização entre amigos, o que em partes explica a grande repercussão do aplicativo no mundo todo (KIM, 2015).

2.3 ÁREAS DE APLICAÇÃO

Ysmar Vianna e outros autore (2013) destacam que no mundo dos negócios, a maneira tradicional de se fazer marketing, criar produtos, prestar serviços e engajar pessoas já não é mais absoluta. Com frequência cada vez maior o uso de gamificação tem sido aplicado por empresas e entidades de diversos segmentos, não só como um atrativo ao seus produtos, mas como alternativas às abordagens tradicionais para os diversos processos do cotidiano empresarial e institucional como treinamentos, familiarização com novas tecnologias e realização de tarefas consideradas tediosas e ou repetitivas.

Apesar do sucesso no meio empresarial, o conceito de gamificação é bastante flexível. De acordo com Zichermann e Cunningham (2011), pode ser aplicado a qualquer contexto que envolvam a motivação e engajamento das pessoas. Por esse motivo, a gamificação está sendo usada em diversos campos da sociedade, como educação, meio ambiente, saúde e bem estar, motivação pessoal, e outros (VIANNA, 2013).

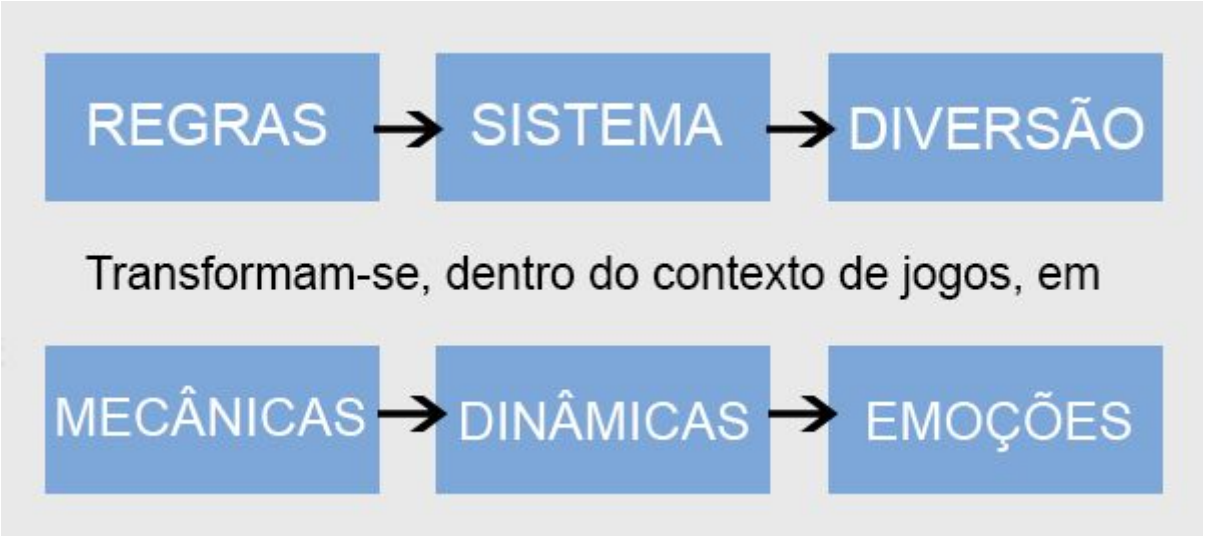
Na educação, diversas instituições de ensino sugerem que a gamificação pode despertar a motivação dos estudantes e fazer com que aprendam de maneira mais efetiva e se envolvam mais com os estudos [23]. Pessoas, e principalmente jovens na fase de estudos, gastam voluntariamente inúmeras horas em frente a um computador, *smartphone* ou *video game* resolvendo problemas que os jogos colocam em seus caminhos (Gee, 2008 apud 23), e com isso, acabam reconhecendo o valor do esforço e desenvolvem diversos atributos como persistência, criatividade, e resiliência ao jogar por tanto tempo (McGonigal, 2011, apud 23). Na educação, a gamificação entra como uma forma de tentar trazer esses

benefícios para as escolas, cursos, e ambientes de aprendizado, que vêm cada vez mais apresentando problemas de engajamento e motivação dos alunos [23].

2.4 ELEMENTOS DE JOGOS

O problema de definir o que é um jogo e quais são seus componentes vem sendo estudado há muito tempo na área de design de jogos. O MDA *framework* (*mechanics*, *dynamics and aesthetic*), elaborado por Robin Hunicke, Marc LeBlanc e Robert Zubek (2001), é um dos modelos mais conhecidos para essa finalidade (Kim, 2015; Zichermann e Cunningham, 2011; VIANNA, 2013) . O MDA divide o jogo em três componentes fundamentais: **regras**, **sistema**, e **diversão**, que sob a ótica do design e desenvolvimento de jogos transformam-se em **mecânicas**, **dinâmicas** e **estéticas** (sendo estética a tradução literal de *aesthetic,* que no contexto de jogos entende-se por emoções relacionadas à diversão).

Figura 1. Os três componentes do MDA framework (fonte: autoria própria)



Fonte: autoria própria

A **estética** representa as reações emocionais que são despertadas no jogador durante uma experiência de jogo (Robin Hunicke *et al.* 2004). Devido a característica dos jogos de ser um objeto de entretenimento, cada uma dessas emoções estão de certa forma ligadas à diversão, ou seja, representam a manifestação da diversão em diferentes formas, como mostra a tabela a seguir.

Sentimento	Descrição
Sensação	diversão em forma de prazer sensorial
Fantasia	diversão relacionada ao faz-de-conta
Narrativa	diversão em forma de drama
Desafio	diversão pela superação de obstáculos
Interação social	diversão gerada por socialização
Descoberta	diversão na forma de dominação
Expressão	diversão como auto-expressão
Submissão	diversão como passa-tempo

As emoções surgem na medida em que o jogador interage com os outros componentes do jogo (Robin Hunicke *et al*. 2014). No xadrez, por exemplo, a diversão na forma de *sensação* pode ser despertada quando uma boa jogada é realizada, enquanto a diversão através do *desafio* pode vir quando o jogador se encontra em uma situação desfavorável e precisa superá-la para vencer, como no xeque-mate.

As **dinâmicas** definem a maneira como os jogadores irão interagir com as mecânicas e como irão se comportar durante o jogo (ZICHERMANN e CUNNINGHAM, 2011). As dinâmicas entram como um elemento externo, são as diversas formas e táticas que os jogadores encontram para vencer ou aproveitar melhor o jogo (Robin Hunicke *et al*. 2014). Por exemplo em poker, existe a dinâmica de *blefar* (que significa mentir, “jogar verde”), que é uma tática usada pelos jogadores para tentar vencer a rodada mesmo com uma combinação de cartas ruim, onde os outros jogadores desistem acreditando que o jogador que está blefando realmente possui uma boa combinação.

Ao desenvolver um jogo, pode-se pensar em dinâmicas que seriam interessantes para se obter os resultados desejados, pois são um importante recurso usado para despertar as emoções (estética). Por exemplo, a emoção de desafio, pode ser estimulada por dinâmicas que exigem jogadas rápidas dos jogadores ou que envolva oponentes. Da mesma forma, o sentimento de amizade e socialização pode ser gerado por uma estrutura que influencie a dinâmica de troca de informação entre jogadores (Robin Hunicke *et al*. 2014).

As **mecânicas** são os componentes funcionais do jogo, ou seja, as regras e tudo aquilo que se pode interagir ou observar (Robin Hunicke *et al*. 2014). Por

exemplo, em um jogo de tiro em primeira pessoa online, as armas, munições e personagens são mecânicas, bem como a velocidade dos tiros e personagens, o dano causado por cada arma, a geografia local, etc. Além dos elementos interativos, como estes mencionados, ainda existem mecânicas que servem apenas como indicadores de progresso ou desempenho, como *rankings*, *status* de missão, placares, mapas, barras de vida, barras de progresso, recursos adquiridos, poder de ataque, poder de defesa, e tudo aquilo que indica ou informa o jogador sobre um determinado contexto.

2.4.1 Elementos de jogos em gamificação

Muitas das preocupações que envolvem a criação de jogos em seu contexto natural (de apenas puro entretenimento), podem ser abstraídas na criação de sistemas gamificados (ZICHERMANN e CUNNINGHAM, 2011). Apesar de não existir na prática nenhuma limitação no uso das mecânicas, dinâmicas e estéticas de jogos em ambientes de gamificação, alguns elementos acabam se destacando. No caso das mecânicas, os autores Zichermann e Cunningham (2011) destacam algumas como sendo de maior relevância, dado a frequência com que aparecem em contextos de gamificação bem como sua efetividade e fácil compreensão, como descrito a seguir.

2.4.1.1 Pontos

Nos jogos, os pontos são a representação numérica de algum recurso ou atributo existente naquele universo. Por ser uma quantia numérica, os pontos são

bastante flexíveis e são uma maneira prática e eficiente de ensinar aos jogadores o que é mais importante dentro do contexto onde estão sendo aplicado (coisas mais importantes valem mais pontos).

O Habitica (HABITICA, 2018) um aplicativo para organização de hábitos e tarefas, utiliza a mecânica de pontos para representar diversos atributos de gamificação dos seus usuários. Na figura 2, onde mostra a tela principal do Habitica, podemos observar que um jogador possui três barras indicando seus pontos de vida, seus pontos de experiência e seus pontos de energia. Esses atributos representam a situação atual do personagem e são modificados na medida que os usuários interagem com o aplicativo, cumprindo ou descumprindo tarefas, praticando bons ou maus hábitos, etc.

Figura 2. tela inicial do aplicativo Habitica

Fonte (HABITICA, 2018)

2.4.1.2 Nível

O nível é uma mecânica que tem como principal objetivo medir progresso e balancear a dificuldade. O nível pode ser usado tanto para o usuário se situar dentro da linha evolutiva do jogo (o usuário evolui de nível) quanto para os desenvolvedores terem mais controle quanto à evolução da dificuldade, complexidade e exposição de novos recursos.

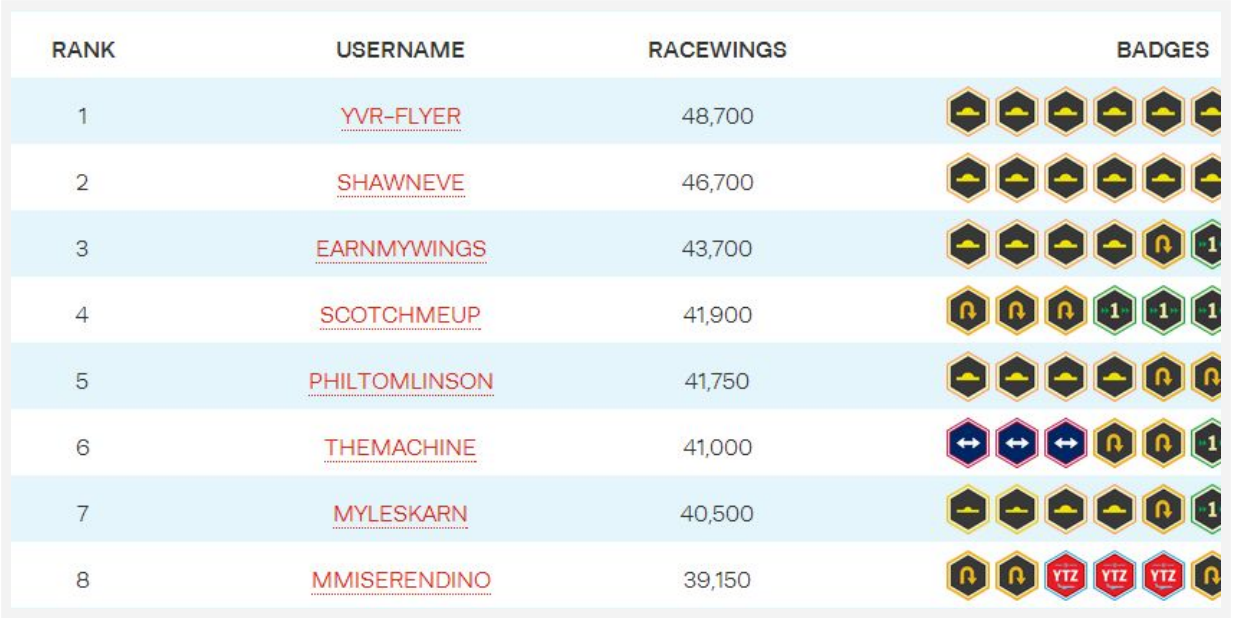
Por exemplo, supondo um sistema de ensino online gamificado, o nível do estudante ou da turma poderia ser usado pelo professor para passar questões mais complexas (dificuldade), detalhar mais o conteúdo (complexidade) ou apresentar novos tipos de atividades (novos recursos).

2.4.1.3 Ranking

O *ranking* é uma lista simples que serve para indicar quem está na frente em relação a algum atributo (ex.: jogadores com maior número de insígnias). Geralmente os *rankings* são usados em contextos competitivos por promoverem a motivação dos jogadores através da disputa pela primeira colocação (uma recompensa em forma de *status*).

A Air Canada, uma das maiores linhas aéreas do canadá usa um sistema de gamificação onde os passageiros podem acumular *racewings* (pontos) e ganhar diferentes medalhas apenas viajando e completando missões e tarefas específicas. Posteriormente os clientes podem trocar os *racewings* por descontos e outras recompensas. A figura 3 mostra o *ranking* dos clientes com maior quantidade de *racewings* e suas *badges* (conquistas).

Figura 3. Ranking dos usuários em recewings da linha aérea Air Canada.



RANK	USERNAME	RACEWINGS	BADGES
1	YVR-FLYER	48,700	
2	SHAWNEVE	46,700	
3	EARNMYWINGS	43,700	
4	SCOTCHMEUP	41,900	
5	PHILTOMLINSON	41,750	
6	THEMACHINE	41,000	
7	MYLESKARN	40,500	
8	MMISERENDINO	39,150	

fonte: (BRANDUNIC, 2013)

2.4.1.4 Insígnias

Também conhecidas como *badges, medalhas,* ou *achievements* nos jogos, as insígnias são uma das mecânicas mais usadas em sistemas de gamificação. São usadas com o intuito de certificar um jogador por ter feito ou alcançado algo importante ou inusitado dentro do jogo, algo muito parecido com as medalhas de guerras.

As pessoas possuem diversos motivos diferentes para gostar das insígnias. Alguns podem gostar pelo *status*, outros gostam da conquista e sensação de reconhecimento, outros gostam de colecionar. As medalhas tem uma forte relação com a parte estética do jogo, podendo despertar sentimentos importantes nos jogadores, tanto de maneira individual como social.

2.4.1.5 Missões

Em jogos, as missões aparecem como um conjunto de tarefas ou objetivos que devem ser alcançados para que um objetivo maior (a missão) seja cumprida. Em gamificação as missões servem principalmente para ensinar os usuários sobre determinados procedimentos mais complexos, que às vezes envolvem diversos passos ou etapas. É uma mecânica perfeita para tutoriais e guias. Utilizar missões e desafios pode ser também de fundamental importância para aumentar a profundidade e o significado do jogo em casos que envolvam algum tipo de narrativa.

2.4.1.5 Customização

A customização nada mais é do que a mecânica e dinâmica que permite que os usuários modifiquem os elementos gráficos e de design do jogo ou sistema. Apesar de não ser tão demandada, esta é uma técnica que pode agregar valor à experiência. Ideias de customização variam desde permitir que o usuário altere a cor de seu plano de fundo até a customização 3D de seu personagem dentro de um contexto narrativo. As redes sociais mais populares, por exemplo, utilizam deste recurso quando permitem que os usuário troquem sua imagem de fundo.

2.4.2 Feedback instantâneo e estímulo

Outra importante característica dos jogos, e que faz parte da essência dos jogos, é o *feedback* instantâneo (ZICHERMANN e CUNNINGHAM, 2011). Os jogos digitais possuem diversos tipos de *feedbacks*, que nada mais são do que informações ou mensagens de destaque que são mostradas aos jogadores à

medida em que o mesmo interage com as regras e elementos do jogos. Essas mensagens podem indicar progresso, conquistas ou qualquer ação relevante dentro do contexto do jogo (ZICHERMANN e CUNNINGHAM, 2011). Por ser dado em tempo real, o *feedback* tem o poder de estimular o jogador de maneira contínua, indicando o caminho correto e incentivando suas boas ações, o que torna a experiência em uma avaliação em tempo real que incentiva e guia o jogador de maneira efetiva (ZICHERMANN e CUNNINGHAM, 2011).

3. ESTADO DA ARTE

Em meio ao grande crescimento das expectativas acerca da gamificação nos últimos anos, diversas *startups* e agências de design foram fundadas com a proposta de oferecer serviços relacionados a gamificação por meio de consultorias, ferramentas e plataformas (Seaborn & Fels, 2015). A técnica tem sido adotada por grandes e pequenas empresas de diversas áreas como tecnologia da informação, educação, saúde, bem estar, e diversas outras aplicações com propósitos particulares.

Hoje, a prática de gamificação se encontra em um estágio mais maduro e menos midiático, com um mercado estável e que continua em crescimento, como mostra uma pesquisa de 2016 realizada pela Markets and Markets, onde é estimado que o valor gerado pela indústria de gamificação pode chegar a US$ 11 bilhões até 2020, um crescimento de mais de 45% em relação a 2014.

Neste capítulo são apresentados alguns casos de uso, divididos em duas partes. A primeira referente-se a produtos e empresas que fazem uso de gamificação. A segunda parte, aborda sobre ferramentas e plataformas de gamificação que oferecem serviços e soluções prontas e que se aproximam um pouco mais da proposta deste trabalho.

3.1 USO DE GAMIFICAÇÃO EM APLICAÇÕES REAIS

Devido ao seu propósito abrangente, a técnica de gamificação tem sido adotada por empresas e instituições de diversos segmentos da sociedade. Esta seção tem por objetivo apresentar aplicações reais de diversas áreas como saúde,

educação, gerência de tarefas e tecnologia da informação, etc., que fazem uso de gamificação em seus produtos.

3.1.1 Gamificação no gerenciamento de tarefas

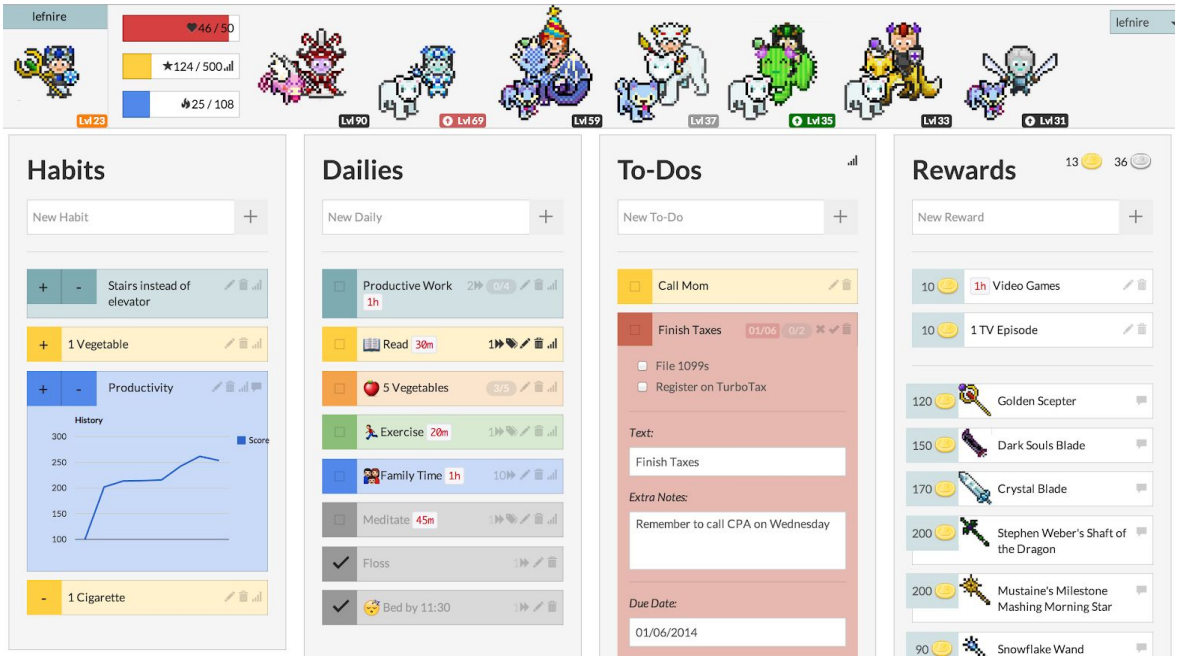
Nos dias atuais das multitarefas é comum ver pessoas fazendo listas, criando eventos na agenda e deixando bilhetes em todo canto para lembrar dos diversos afazeres do dia-a-dia, do trabalho e dos estudos. Também é comum que, diante de tantas coisas por fazer, as pessoas acabem ficando sobrecarregadas demais e deixando de lado esses afazeres. Com o propósito de amenizar esse problema, aplicativos como EpicWin, Habitica e Chore Wars utilizam gamificação para tornar a realização de tarefas mais divertida e motivar seus usuários a não desistirem de seus objetivos.

3.1.1.2 Habitica

Também conhecido como HabitRPG, o Habitica é uma aplicação web com versão para mobile, open source, designada ao gerenciamento de tarefas. O jogador pode criar tarefas e hábitos e colocar em uma das três categorias: Hábitos, tarefas diárias e tarefas a fazer (figura 4). Ao marcar uma tarefa como feita, o jogador ganha pontos para evoluir de nível e uma quantia de ouro, que mais tarde pode ser gasto com itens cosméticos do jogo. Uma das características interessantes do Habitica é que assim como em diversos games, o personagem possui pontos de vida, e cada vez que uma tarefa diária não é cumprida ou um hábito ruim é praticado, o personagem perde pontos vida, simbolizando então uma luta entre a pessoa e seus afazeres e hábitos.

Dentro do sistema do Habitica, o jogador pode se juntar a grupos específicos com outros jogadores, trocar mensagens, enfrentar monstros coletivamente e interagir com personagens criados pelo jogo, que solicitam ajuda aos usuários para se defenderem de vilões. Para derrotar os vilões, os jogadores precisam causar dano completando suas tarefas e realizando bons hábitos.

Figura 4. Painel principal do Habitica onde são mostrados os hábitos, tarefas diárias, afazeres e recompensas.



Fonte: (HABITICA, 2018)

3.1.2 Gamificação na saúde e bem estar

Gamificação na área de saúde e bem estar tem se tornado bastante popular nos últimos 5 anos (KIM, 2015). A prática de exercícios físicos bem como o controle de consumo de alimentos exige uma motivação que por vezes as pessoas não conseguem encontrar por conta própria. Pensando nisso, pequenas e grandes

empresas relacionados ao ramo estão incorporando gamificação em seus produtos ou simplesmente desenvolvendo sistemas específicos para isso, como é o caso do Nike +, um app de corrida desenvolvido pela Nike para estimular as pessoas a praticar exercícios.

3.1.2.1 Nike + running

Correr e caminhar são exercícios físicos presentes no dia-a-dia de muita gente. A Nike, uma empresa de roupas e calçados de esporte, desenvolveu o aplicativo para celular Nike+ running, que usa o GPS do dispositivo para registrar a quilometragem das corridas e caminhadas dos usuários e mostrar diversas informações como batimentos cardíacos, velocidade, entre outros. O aplicativo utiliza a gamificação permitindo que o usuário crie desafios e tente superá-los para ganhar pontos e medalhas que poderão ser visualizadas por outros usuários do aplicativo. Além disso, uma das grandes atrações é a possibilidade de criar desafios coletivos, onde amigos, familiares e colegas podem somar esforços para superar, promovendo a socialização e engajamento coletivo dos usuários, resultando em corridas e caminhadas com mais adesão.

Figura 5. Telas de compartilhamento de atividades, estatísticas de corrida e desafios do aplicativo Nike + running.

Fonte: (ENGADGET, 2013)

3.1.3 Gamificação na educação e ensino

Na área de educação, a gamificação tem o papel de enriquecer o processo de aprendizado introduzindo a maneira de pensar dos jogos, incentivando os estudantes a partir de *feedbacks* e recompensas mais precisas sobre seus esforços, promovendo maior engajamento e motivação em tempo real (McGonigal, 2012).

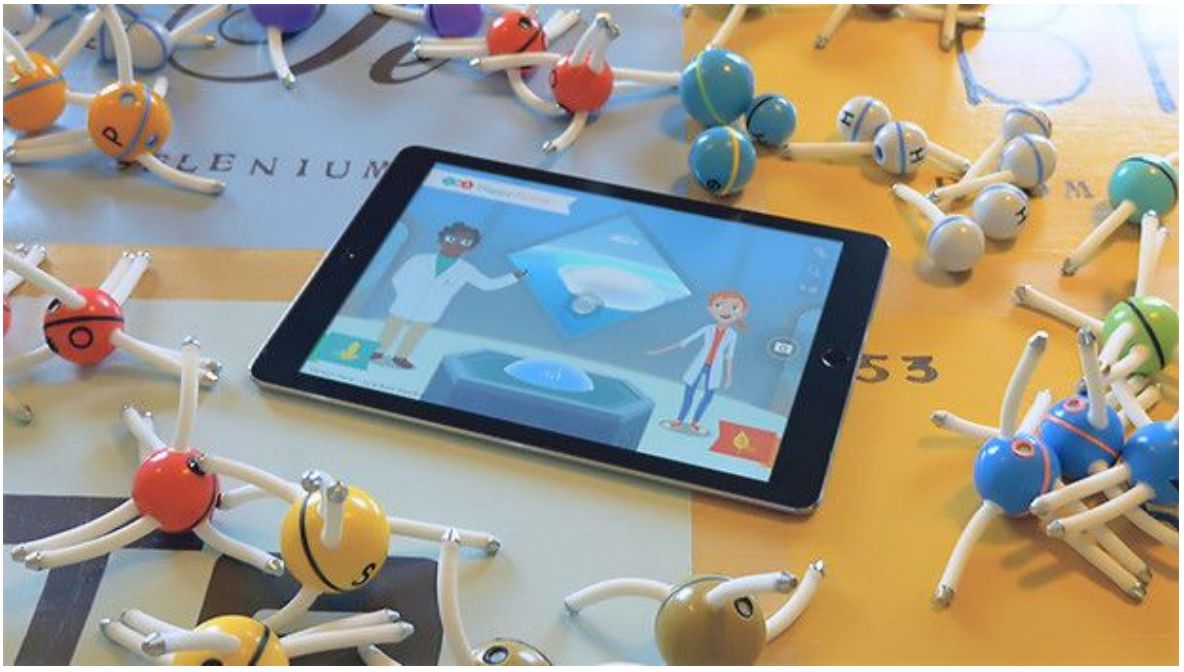
Apesar de ser uma área onde a pesquisa e discussão sobre o assunto ainda é bastante movimentada, diversos aplicativos estão usando gamificação em seus produtos para melhorar a experiência de seus usuários, principalmente na área de *e-learning* como é o caso do Happy Atoms e o do Duolingo.

3.1.3.1 Happy Atoms

O Happy Atoms é uma ferramenta de aprendizado que combina um conjunto de peças físicas de modelagem com um aplicativo digital que interage com esses

modelos (figura 6). Sua proposta é ensinar combinação atômica, um assunto importante da química, aos alunos da 4° série em diante. De acordo com a empresa criadora, a maneira como a química é ensinada nos dias atuais desencoraja os estudantes a explorarem e descobrirem. Com o Happy Atoms, os estudantes participam diretamente do aprendizado. Com o auxílio de um guia, os estudantes constroem as moléculas físicas que se conectam através de imãs [12]. O segundo passo é escanear a molécula recém criada com o aplicativo, que irá identificar a molécula e fornecer diversas informações sobre a mesma. A princípio são 150 moléculas para descobrir e diversos desafios e tarefas que já vem com o aplicativo.

Figura 6. Peças e aplicativo do Happy Attoms



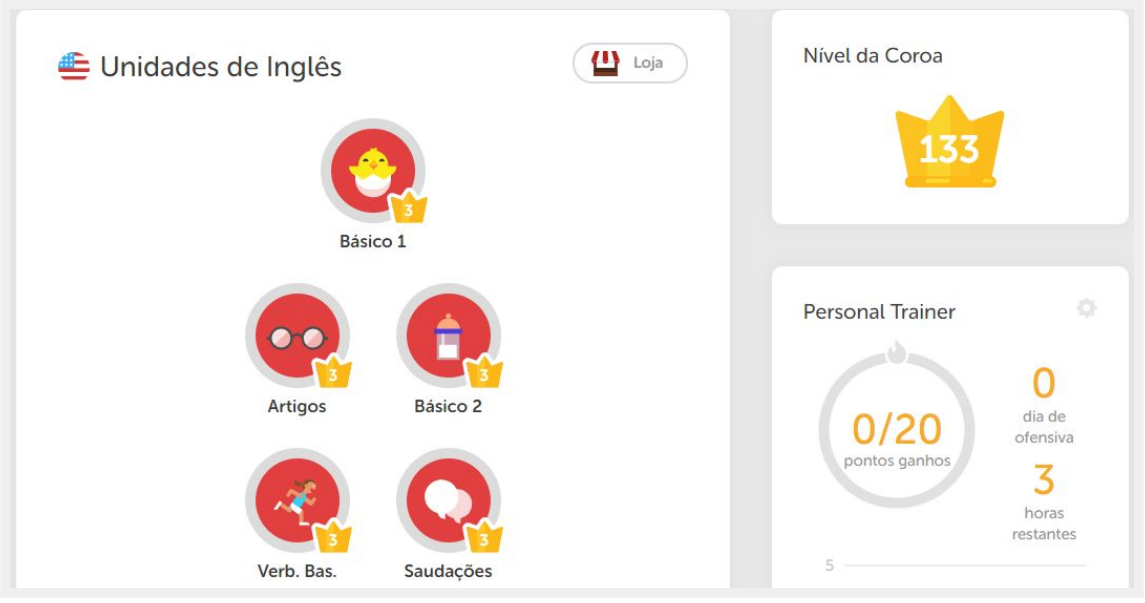
Fonte:(TECHCRUNCH, 2016)

3.1.3.2 Duolingo

O Duolingo é uma plataforma grátis para o aprendizado de idiomas criada por Luis Von Ahn e seu aluno Severin Hacker. A ideia é que os usuários aprendem novos idiomas ao mesmo tempo em que ajudam a traduzir diversos documentos. O Duolingo foi lançado em 2012 com mais de 300.000 usuários. Em 2016 a plataforma já oferecia mais de 59 cursos em 23 idiomas diferentes. O Duolingo faz uso de gamificação para motivar e engajar os estudantes usando alguns elementos de jogos e a filosofia dos jogos, promovendo socialização e competição entre os estudantes. Dentre os elementos utilizados pelo Duolingo, destacam-se os níveis, medalhas, conquistas, ranking e pontos em diversas formas.

A figura 7 mostra o perfil de um usuário do Duolingo, onde é possível ver o nível atual em cada umas das categorias de palavras no idioma inglês, além do nível da coroa, uma soma de todo seus níveis de conhecimento em cada categoria. Na mesma tela, também aparecem os desafios diários e o progresso do estudante.

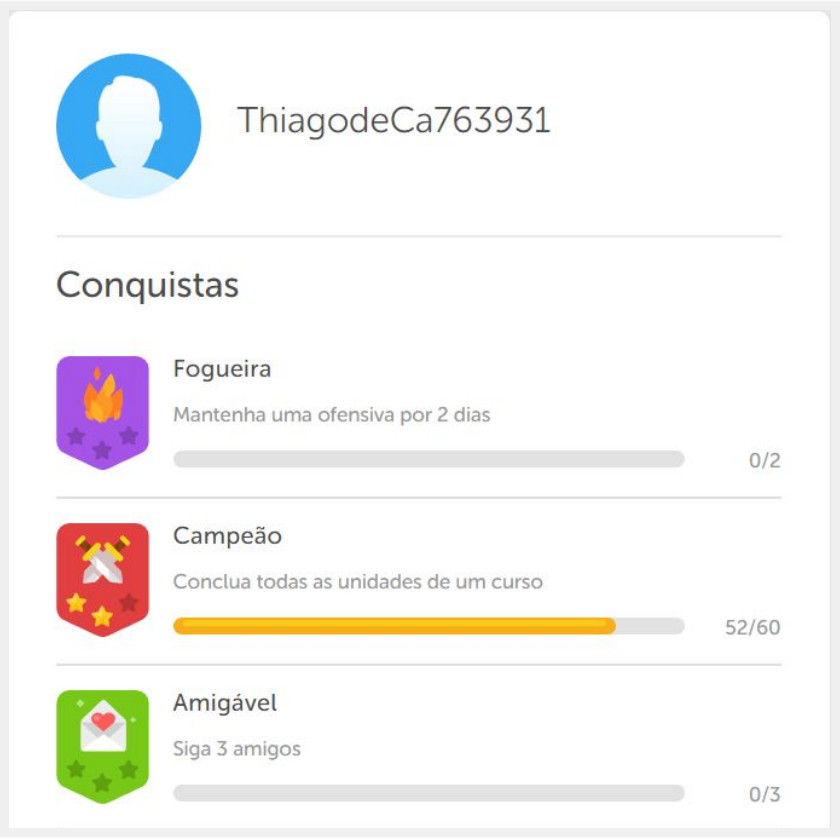
Figura 7. Perfil do usuário na plataforma Duolingo



Fonte (DUOLING, 2018)

Além dos níveis e desafios diários, o Duolingo também aposta num sistema de conquistas relacionadas tanto ao aprendizado quanto à frequência e dedicação do usuário ao longo do percurso de aprendizado. A figura 7 mostra a tela de atuais conquistas do usuário e seu progresso.

Figura 7. Conquistas do usuário da plataforma Duolingo



Fonte (DUOLING, 2018)

3.2 FERRAMENTAS DE GAMIFICAÇÃO

A pesquisa de mercado feita pela Markets and Markets em 2016 revela que a indústria em torno de gamificação ainda está crescendo. Seja com empresas mais

consolidadas ou *startups*, a variedade de ideias e soluções para gamificação ainda não chegou em sua maturidade completa.

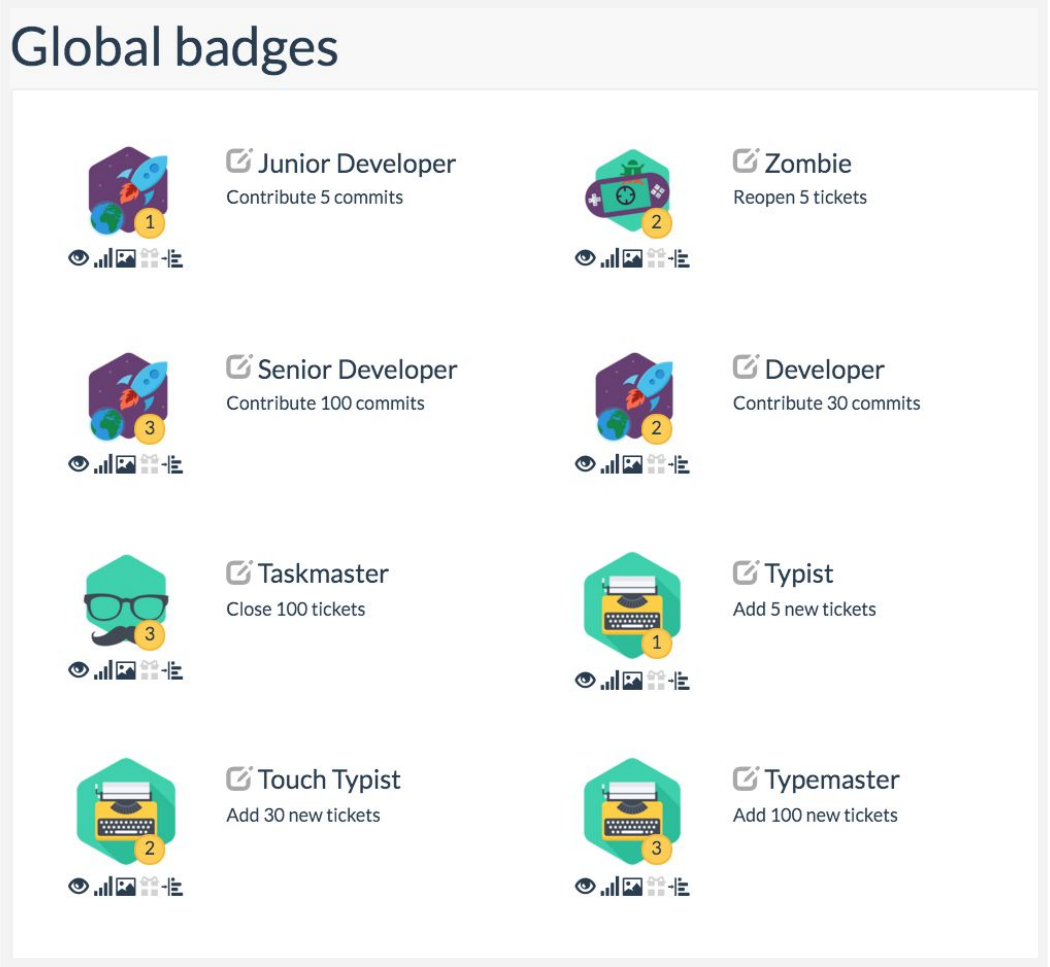
Essa seção tem por objetivo abordar sobre ferramentas, plataformas, engines, etc., de gamificação que se assemelham ou possuem características próximas à proposta do trabalho, mostrando para cada uma, qual o seu propósito e de que forma trabalha com os elementos de gamificação.

3.2.1 GetBadges

O GedBadges é uma plataforma de gamificação voltada para equipes de desenvolvimento de software. A proposta da ferramenta é promover um ambiente online mais descontraído e gratificante para as equipes de desenvolvimento através de elementos de jogos. Uma de suas características mais importante é a integração com diversas outras ferramentas de desenvolvimento de software populares como Trello, Slack, Jira, RedTime, GitHub, etc., que fornecem a movimentação de atividades de cada usuários servindo como base para medir e rastrear o progresso da equipe, seja resolvendo pequenos bugs ou implementando grandes pacotes.

A partir do rastreamento do progresso, os desenvolvedores são recompensados com insígnias personalizadas pelo próprio time. A plataforma conta também com *feedbacks* coletivos e notificações, além de outras características de gamificação voltadas para a competitividade, como os *rankings*.

Figura 8. Amostra de insígnias criadas para a plataforma GetBadges



Fonte (GETBADGES, 2018)

Além dos clássicos elementos de gamificação, a GetBadges também conta com opção de criar desafios lúdicos em forma de monstros que representam uma tarefa ou projeto. Seguindo essa filosofia, ao eliminar pendências e entregar tarefas, o monstro é atingido e perde vida, enquanto que atrasos e outras faltas de comprometimento acabam regenerando o monstro.

A figura 9 mostra uma representação da batalha virtual entre o time de desenvolvimento e o inimigo e as possíveis recompensas em forma de elementos

lúdicos. Ao gerar resultados positivos, o desenvolvedor ganha experiência e conquistas, que eventualmente o fazem subir de nível e causar danos aos monstros. Em contrapartida, resultados negativos acabam ajudando os monstros a sobreviverem.

Figura 9. representação das recompensas em relação às atividades realizadas na plataforma GetBadges.

Fonte (GETBADGES, 2018)

3.2.2 Badgville

Badgville é uma das plataformas de gamificação mais bem sucedidas no meio corporativo. Fornece uma estrutura baseada em gamificação que atua em diversos segmentos empresariais, como interação social entre equipes, performance de vendas, treinamentos, gerência e suporte ao cliente. Tem uma forte atuação na mudança de comportamento de funcionários em relação ao uso das ferramentas da empresa, promovendo aspectos motivacionais e atuando como um guia.

Além de ser usada para guiar e motivar os funcionários no uso das ferramentas, a plataforma atua em todo o ciclo de produção fornecendo meios para definição de objetivos como também a análise do sucesso e o alcance da plataforma com estatísticas e gráficos.

Suas principais características de gamificação incluem pontos, medalhas, missões, progresso, recompensas sociais, quadros de liderança e notificações de atividades.

Figura 10. Tela de perfil do usuário na plataforma Badgville

Fonte (BADGEVILLE, 2018)

3.2.3 Gengine

A Gengine, também reconhecida como gamification-engine, é um software open source (MIT) para integração de gamificação em projetos de software [23].

Diferente das plataformas de gamificação mais comuns no mercado, a Gengine não disponibiliza nenhum layout gráfico, estatística ou interface de gerenciamento, mas sim *web services* que podem ser acessados por aplicações via API.

Características e funcionalidades disponíveis na Gengine:

- Conquistas conforme evolução de nível
- Conquistas conforme objetivos são alcançados
- Acompanhamento de progresso e objetivos únicos
- Quadros de liderança, sistema de *ranking*
- Conquistas podem ser alcançadas imediatamente ou através de verificação diárias, mensal ou anual
- É independente de fuso horário
- Alerta em escopo social (ex.: alcançar maior pontuação entre amigos)
- Alerta em escopo geográfico (ex.: alcançar maior pontuação na cidade)
- Regras podem ser definidas em python através de variáveis (ex.: o nível atual de um jogador)
- Definição personalizada de propriedades de conquistas e recompensas
- Definição de idiomas e traduções de maneira personalizada
- pré-requisitos e pós condições de conquistas

4. PROPOSTA E DESENVOLVIMENTO DO TRABALHO

Na introdução e nos capítulos anteriores, a técnica de gamificação e os problemas e preocupações acerca de sua implantação e manutenção em diversos contextos foram levantados. Apesar de ser necessário a programação (desenvolvimento de software) da gamificação em qualquer sistema digital, a maior preocupação fica na elaboração de uma boa estratégia que faça sentido e possua significado aos usuários. Levando em consideração esses aspectos, este trabalho tem como proposta o desenvolvimento de uma plataforma que facilite a introdução de gamificação em sistemas *web*, *desktop* e *mobile* com o objetivo de tirar das mãos do criador a responsabilidade e o esforço de desenvolver a solução técnica, abrindo margem para elaboração do plano estratégico e diminuindo o tempo e custo total do projeto.

Neste capítulo serão apresentados a definição e o escopo do projeto, sua arquitetura, elementos, funcionalidades e como esses recursos interagem entre si.

4.1 PREMISSAS E ESPECIFICAÇÃO DE SOFTWARE

Levando em consideração os estudos realizados em torno das outras soluções de mercado e dos objetivos deste trabalho, foi necessário criar uma solução multiplataforma, de fácil configuração, integração e entendimento, que fosse também escalável e modelada de forma que facilitasse a criação de novos recursos no futuro. Tendo estes requisitos e o conceito de gamificação em vista, a primeira tarefa foi definir quais mecânicas de jogos seriam suportadas pela plataforma e como elas seriam atribuídas aos usuários finais, o que resultou na criação de dois

conceitos chaves pelo qual toda a plataforma se sustenta: **os elementos de gamificação** e as **regras de gamificação**.

4.1.1 Elementos de gamificação

Para este trabalho, foram adotados cinco elementos de gamificação, os quais representam uma ou mais mecânicas de jogos:

- Pontos de experiência
- Nível
- Insígnias
- Atributos
- Rankings

Com exceção do elemento atributo, que é apenas uma representação abstrata de qualquer mecânica numérica (ex.: pontos de vida, ouro, fama, reputação, etc.), todos os outros representam uma das mecânicas apresentadas na seção 2.5.1, faltando apenas as missões e customização, que não entram no escopo. Cada um desses elementos são melhor apresentados individualmente na seção 4.2.1.2.

4.1.2 Regras de gamificação

Para que a plataforma pudesse funcionar de maneira automática após sua configuração, foi necessária a criação do conceito de regras de gamificação. Essas regras nada mais são do que pré-condições atreladas ou não a ações que precisam ser cumpridas em determinado contexto para que os elementos de gamificação anteriormente mencionados sejam atribuídos aos usuários finais. Exemplos hipotéticos:

- ”Toda vez que um usuário compartilhar uma notícia, ele receberá 200 pontos de experiência”;
- “Ao fazer 5 comentários, o usuário ganhará 100 pontos de atributo fama”;
- “Do nível 1 ao 10, sempre que atingir 1000 pontos de experiência, o usuário subirá de nível”;
- “Ao acumular 20000 pontos no atributo fama, o usuário receberá a insígnia *um astro do rock*”.

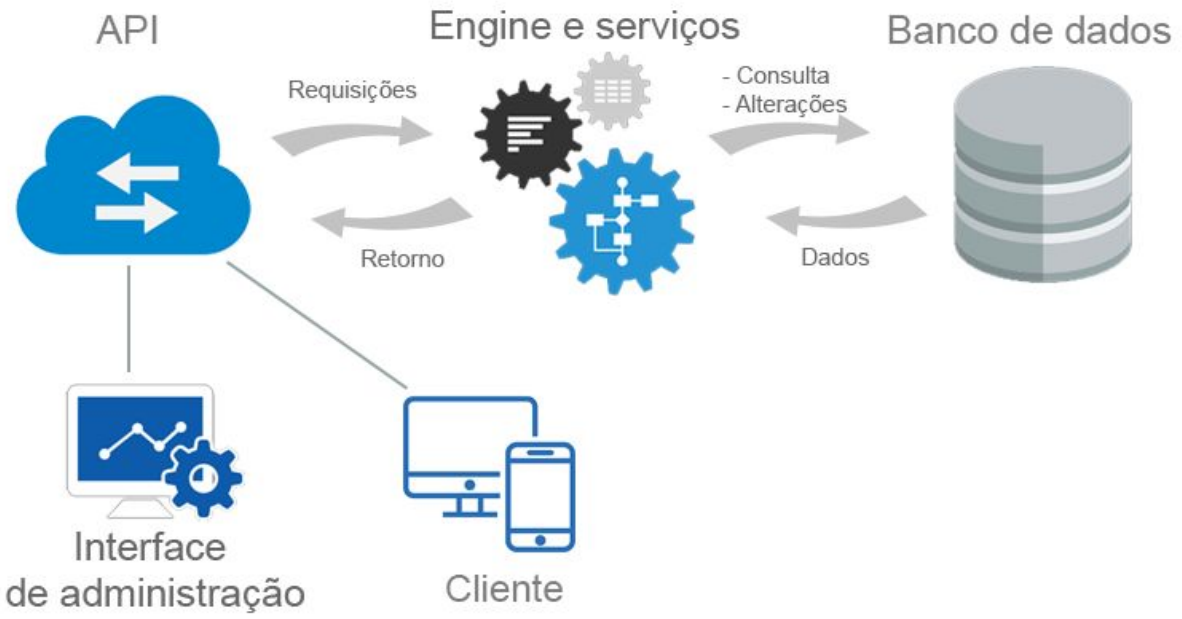
Em outras palavras as regras servem para definir quando e como que os usuários devem ganhar pontos de experiência, pontos de atributos, subir de nível, conquistar uma insígnia, etc. Para cada elemento de gamificação, foi elaborado um tipo de regra específica, cuja criação e configuração ficam a cargo do responsável pelo ambiente de gamificação. Cada tipo de regra é apresentada com detalhes na seção 4.2.1.3.

4.1.3 Componentes da plataforma

Uma vez definidos os conceitos principais, foi necessário planejar uma estrutura para criar, gerenciar, processar e armazenar os elementos e regras de gamificação mencionados anteriormente. Para alcançar esses objetivos, foi definido que a plataforma seria composta de cinco componentes principais: (1) uma engine, que é responsável pelo processamento e validação das regras de gamificação; (2) uma central de serviços, que é por onde todos os recursos da plataforma são criados e gerenciados; (3) uma API, que serve de via de comunicação entre o sistema-cliente, a engine, e os serviços; (4) uma estrutura de banco de dados, onde

os elementos, as regras e os usuários que farão parte do ambiente de gamificação serão armazenados; e (5), uma interface de administração da plataforma. A figura 11 mostra uma representação dos cinco componentes e suas respectivas interações

Figura 11. Arquitetura e representação dos componentes da plataforma



Fonte (Autoria própria)

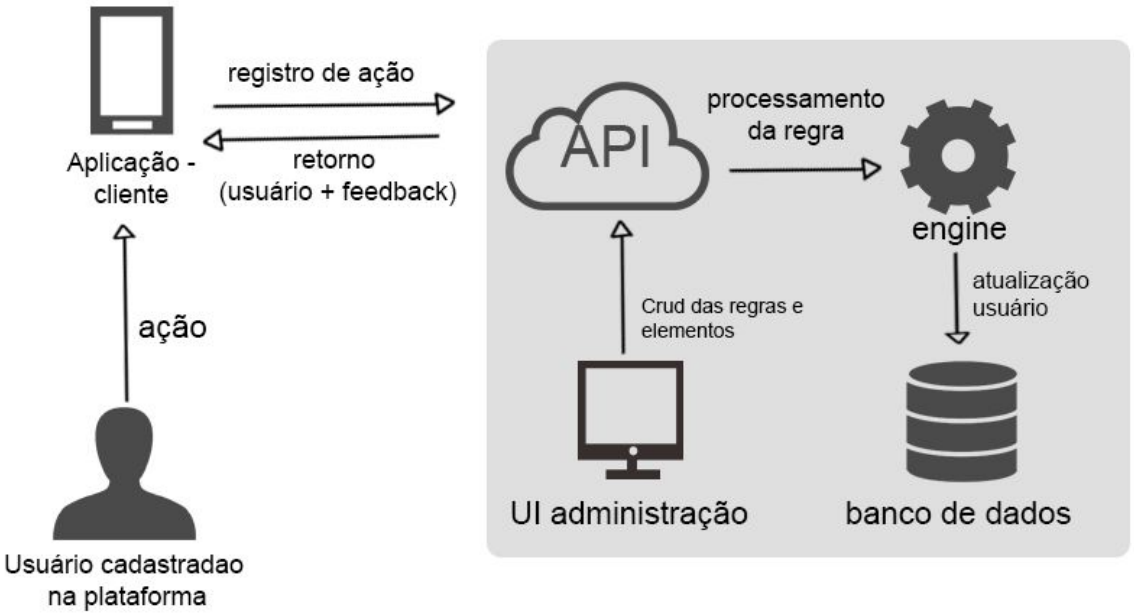
4.2 DEFINIÇÃO DO FUNCIONAMENTO DA PLATAFORMA

O primeiro passo é baixar, instalar e configurar a plataforma em seu local de jogos e regras de gamificação desejáveis, o que pode ser feito via interface de administração ou API, o desenvolvedor do ambiente gamificado precisa popular o banco de dados da plataforma com os usuários que farão parte do sistema. Como é mostrado mais para frente na sessão 4.2.1.1, esses usuários são apenas uma representação dos usuários originais do sistema-cliente.

Após esta etapa, o desenvolvedor precisa acessar o código fonte do seu sistema, seja ele um site, um app, ou outro programa e fazer a integração com a plataforma. Essa integração se dá por meio de registro de atividades dos usuários, que são feitos via requisições HTTP para a API da plataforma, através do seguinte *endpoint*: *http://endereçodoservidor/api/event-rule/{idUsuario}/{idRule}*, onde, o {idUsuario} é o identificador do usuário na plataforma e o {idRule} é o identificador da regra que deve ser processada, que nesse caso também representa um evento ou atividade realizada pelo usuário. Dessa forma, por exemplo, se foi criada uma regra para recompensar usuários que fazem *login* diariamente, é necessário colocar essa chamada de requisição na parte do código onde é efetuado o *login*, especificando corretamente na chamada qual a regra e qual o usuário que efetuou *login*.

Ao receber uma requisição do cliente, com informações do usuário e a regra em questão, a engine irá registrar a atividade no banco de dados e fará o processamento de dados para verificar, através das pré-condições, se a regra foi concluída e atribuir as devidas recompensas ao usuário, retornando para o cliente um *feedback* contento o usuário atualizado e tudo que foi atribuído a ele a partir da ação. Com essas informações, o desenvolvedor programar seu sistema para dar o *feedback* adequado ao usuário que está utilizando sua aplicação, seja através de notificações, *popups*, mensagens, atualizações gráficas, etc. A figura 12 mostra de maneira sucinta toda a interação, desde a ação do usuário até o processamento completo da regra.

Figura 12. Representação do fluxo de processamento de regras da plataforma



Fonte: (Autoria própria)

4.3 ESPECIFICAÇÃO DOS COMPONENTES

Este capítulo tem como objetivo apresentar com mais detalhes cada um dos componentes citados anteriormente. Por definição, a interface de administração não possui uma sessão específica, mas é explorada junto com os outros componentes e recursos conforme estes são apresentados.

4.3.1 Engine e web services

Apesar de não ter uma definição clara, uma engine pode ser visto como um programa para realizar tarefas específicas que envolvam o processamento ou transformação de dados de entrada em dados de saída com outro significado, propósito ou formato. A engine proposta neste trabalho é um dos principais

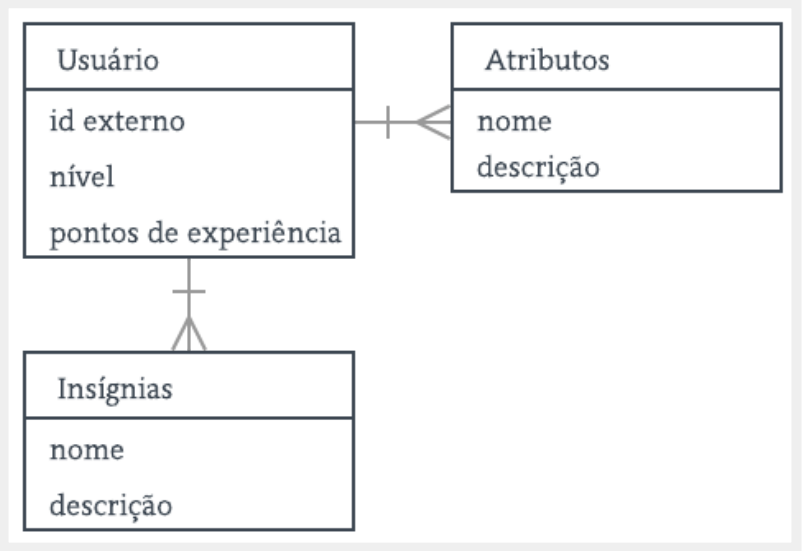
componentes do projeto. É a partir dela que as regras de gamificação são validadas e processadas e os usuário são recompensados. A engine é formada por três recursos principais: os usuários, as regras e os elementos de gamificação, cujo gerenciamento pode ser feito através dos *web services* disponibilizados pela API e também via interface de administração.

4.3.1.1 Usuários

A plataforma possui seus próprios usuários. Todas as interações da engine é feitas com esses usuários exclusivamente. Contudo, o vínculo dos usuários da plataforma com os do cliente, caso seja necessário, pode ser feito através de uma chave externa prevista na criação dos mesmos, como mostra a figura 13.

Outro princípio é que os usuários possuem apenas informações referentes a gamificação e ao funcionamento da engine, como pontos de experiência, nível, insígnias, atributos, etc, não possuindo portanto, qualquer tipo de informação pessoal, os quais ficam a cargo do próprio cliente, na medida de sua necessidade e preservando, assim, aspectos de privacidade dos usuários.

Figura 13. Modelo representativo do usuário e suas informações



Fonte (autoria própria)

Ao criar um novo usuário, todos os atributos criados anteriormente na plataforma são associados a ele automaticamente. As insígnias só podem ser atribuídas aos usuários a partir das regras de gamificação ou manualmente via API. A figura 14 mostra o formulário de criação de usuário na interface de administração com os seguintes campos:

- Nome: Opcional, serve apenas para melhor identificação dos usuários em consultas.
- ID externo: Uma chave que representa o usuário da base real do cliente, pela qual é feito o vínculo entre essas duas instâncias de usuário.
- Nível inicial: determina em que nível o jogador começa.
- Ativo: Se marcado, o usuário já estará ativo no sistema assim que for criado.

Figura 14. Formulário de criação de novo usuário no painel administrativo

Fonte (Autoria própria)

4.3.1.2 Elementos de gamificação

Como visto na seção 2.5.1, existem diversos tipos de elementos de jogos frequentemente encontrados em sistemas de gamificação. Ao completar tarefas e realizar ações, os usuários da engine eventualmente serão recompensados com pontos de experiência, níveis, insígnias e outros atributos. A essas recompensas, chamamos de elementos de jogos.

Neste trabalho, o foco foi em cima dos elementos mais populares, essenciais em praticamente todas as abordagens de gamificação, como visto na seção 4.1.1 e que são apresentados com mais detalhes a seguir.

4.3.1.2.1 Pontos de experiência

Os pontos de experiência são um valor numérico que serve tanto para representar o valor de uma ação - ações mais importantes valem mais pontos -, quanto para representar o progresso do jogador, uma vez que normalmente se faz uso dos pontos de experiência para determinar se o jogador avançou de nível. Por definição, neste trabalho, os pontos de experiências são um atributo nativo do usuário, que começa com zero. Ainda que seja um dos atributos mais frequentes, nem toda abordagem de gamificação utiliza pontos de experiência, e portanto, é aqui tratado de forma opcional. Em casos onde não se use os pontos de experiência para nada, basta apenas ignorar completamente essa recompensa na hora de criar as regras.

4.3.1.2.2 Nível

Como abordado na seção 2.5.1.2, o nível nada mais é do que um valor numérico para representar o progresso do usuário. Por definição, assim como os pontos de experiência, no sistema proposto os usuários possuem nativamente um nível associado, que começa no um. Para subir de nível, o jogador precisa acumular uma quantidade de pontos de experiência, que é especificado pelo desenvolvedor usando as regras de nível, como é explicado no capítulo de regras, mais a frente. O nível também é um elemento opcional, assim como os demais. Caso a abordagem de gamificação adotada não faça uso do nível, basta ignorar este recurso ao criar as regras.

4.3.1.2.3 Insígnias

Também chamadas de *badges*, *achievements* ou medalhas, as insígnias representam uma conquista ou marca atingida dentro do jogo, ou no caso deste trabalho, em ambientes de gamificação. A engine proposta trabalha com dois tipos de insígnias: as relacionadas a eventos e as relacionadas a acúmulo de atributos.

As **insígnias de eventos** são recompensas dadas aos jogadores por realizarem determinada ação uma ou mais vezes. Por exemplo, uma insígnia que marca a conquista de 50 comentários em uma plataforma de ensino.

Seguindo a mesma lógica, as **insígnias de atributos** são dadas aos jogadores ao acumularem determinada quantidade de algum atributo existente na plataforma. Por exemplo, supondo que exista um atributo chamado pontos de participação, poderia ser criada uma insígnia que representa a marca de 10000 pontos de participação. Independente do tipo de insígnia, tanto a quantidade necessária de ações, quanto o valor a ser acumulado são definidos via criação das regras de insígnias (descritas mais adiante).

Figura 15. formulário de criação de nova insígnia no painel administrativo

Fonte (Autoria própria)

4.3.1.2.4 Atributos

Os jogos possuem uma grande gama de elementos numéricos. Pontos de vida, energia, velocidade, força, ouro, riquezas, e uma infinidade de outros elementos são encontrados com frequência nos jogos em geral. Neste trabalho, a proposta é trazer esses elementos para a gamificação de maneira genérica, criando apenas um elemento chamado atributo. Na configuração e criação dos elementos, é possível criar quantos atributos for necessário. Todos usuários, ao serem criados, já são associados a todos os atributos existentes, começando com zero pontos em cada. Posteriormente, caso seja necessário, esses atributos podem ser desabilitados para usuários específicos.

Um atributo possui um nome e uma descrição apenas, como pode ser visto na figura 16 do formulário de criação dos atributos, que está disponível via na interface de administração.

Figura 16. Formulário de criação de atributos na interface de administração

Fonte (Autoria própria)

4.3.1.3 Regras

As regras de gamificação definem as condições para que os elementos de gamificação vistos anteriormente sejam atribuídos aos usuários. As regras são classificadas por tipos, onde o tipo depende do elemento de jogo para qual a regra é criada. Para cada tipo de regra são definidos diferentes tipos de condições, como é mostrado no item específico de cada regra a seguir.

4.3.1.3.1 Regras de experiência

As regras de experiência são criadas para o elemento pontos de experiência. São úteis para quando se deseja recompensar uma determinada ação do usuário

com apenas pontos de experiência. Ex.: Ao efetuar *login*, o usuário é recompensado com 100 pontos de experiência.

A figura 17 mostra o formulário de criação de uma nova regra de experiência, cujo os campos são:

- Nome: nome da regra, serve apenas para identificação visual
- Recompensa em XP: define quantos pontos de experiência o usuário irá receber quando esta regra for completada.
- Vezes para completar: este campo se refere à quantidade de vezes que o usuário terá que realizar a ação ou atividade em questão para completar a regra e receber os pontos de experiência. Por exemplo,
- Ativar: se marcado, a regra entra em vigor assim que criada.
- Regra cíclica: esta condição define se a regra continua a ser aplicada para um usuário que já tenha completado uma vez.

Figura 17. Formulário de criação de regras na interface de administração

Fonte (Autoria própria)

4.3.1.3.2 Regras de nível e recompensas por nível

Ao criar uma regra de nível, o desenvolvedor decide quantos pontos de experiência o usuário precisa obter para subir de nível dentro de um intervalo de níveis (ex.: do nível 1 ao 10 é necessário 100 pontos de experiência), podendo criar, portanto, diversas regras para abranger todos os níveis e intervalos desejáveis. Quando o usuário atingir o nível máximo, a regra com maior intervalo é usada.

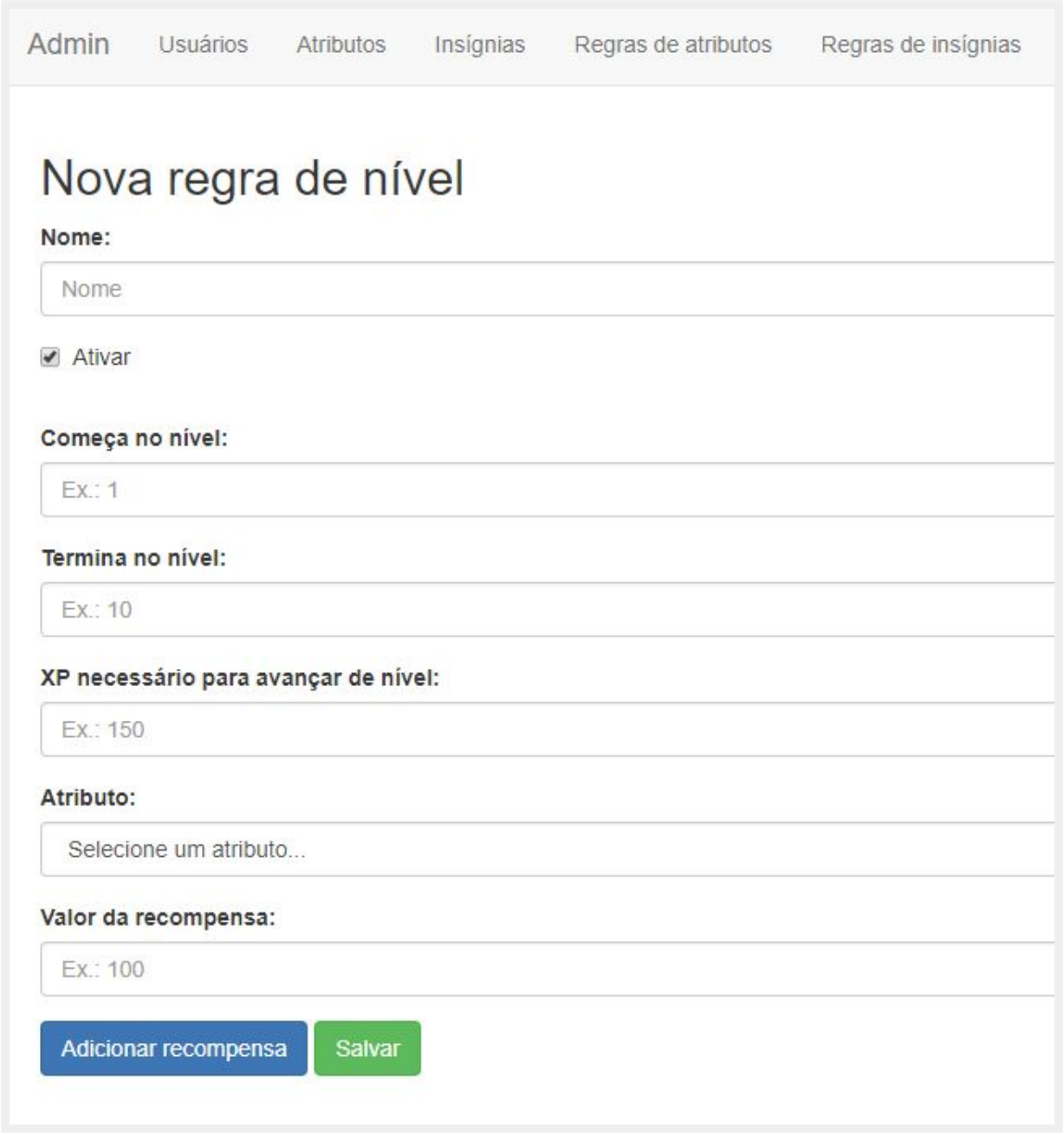
Na criação das regras de nível, também é possível definir recompensas que serão atribuídas ao usuário quando este subir de nível. Contudo, essa recompensa é limitada apenas aos atributos. Por exemplo, supondo um cenário onde os usuários

possuem pontos de vida (um atributo), seria possível estipular uma regra de nível que forneça uma quantidade N de pontos de vida toda vez que o usuário passar de nível dentro daquela margem. Esta regra não é obrigatória e também não está limitada a apenas um atributo por nível, podendo ser atribuídos diversos atributos e em quantidades diferentes.

A figura 18 mostra o formulário de criação de regra de nível no painel administrativo, contento dos seguintes campos:

- Nome: nome da regra, serve apenas para identificação visual
- Ativar: se marcado, a regra entra em vigor assim que criada
- Começa no nível: estipula a partir de que nível esta regra passa a valer
- Termina no nível: estipula a partir de que nível esta regra passa a não deixar de valer
- XP necessária para passar de nível: determina quantos pontos de experiência o usuário precisa acumular dentro dessa faixa para subir de nível.
- Atributo: Esse é um campo opcional e diz respeito às recompensas que serão dadas ao usuário quando este subir de nível. Neste campo é definido qual o atributo que será atribuído.
- Valor da recompensa: determina a quantidade do atributo selecionado que será dado como recompensa ao usuário.

Figura 18. Formulário de criação de regras de nível na interface de administração



Fonte (Autoria própria)

4.3.1.3.3 Regras de atributo

Ao realizar uma ou mais ações, os usuários podem receber pontos em um ou mais atributos existentes. As regras de atributo funcionam apenas com a condição

de eventos, ou seja, a única forma de ganhar atributo é realizando ações, assim como nas regras de experiência. Por exemplo, supondo um sistema de aprendizado onde os usuários possuam o atributo sabedoria, poderia ser criado uma regra que aumentasse a sabedoria dos usuários em N pontos sempre que estes marcassem um texto como lido.

A figura 19 mostra o formulário de criação de uma nova regra de atributo, cujo os campos são:

- Nome: nome da regra, serve apenas para identificação visual
- Recompensa em XP: define quantos pontos de experiência o usuário irá receber sempre que realizar a ação.
- Vezes para completar: este campo se refere à quantidade de vezes que o usuário terá que realizar a ação ou atividade em questão para completar a regra e receber os pontos de experiência.
- Ativar: se marcado, a regra entra em vigor assim que criada.
- Regra cíclica: esta condição define se a regra continua a ser aplicada para um usuário que já a tenha completado uma vez.
- Atributo: determina o atributo que será recompensado quando as condições forem satisfeitas.
- Valor da recompensa: determina a quantidade do atributo selecionado que será dada como recompensa ao usuário.

Figura 19. Formulário de criação de regras de atributo na interface de administração

Fonte (Autoria própria)

4.3.1.3.4 Regras de insígnias

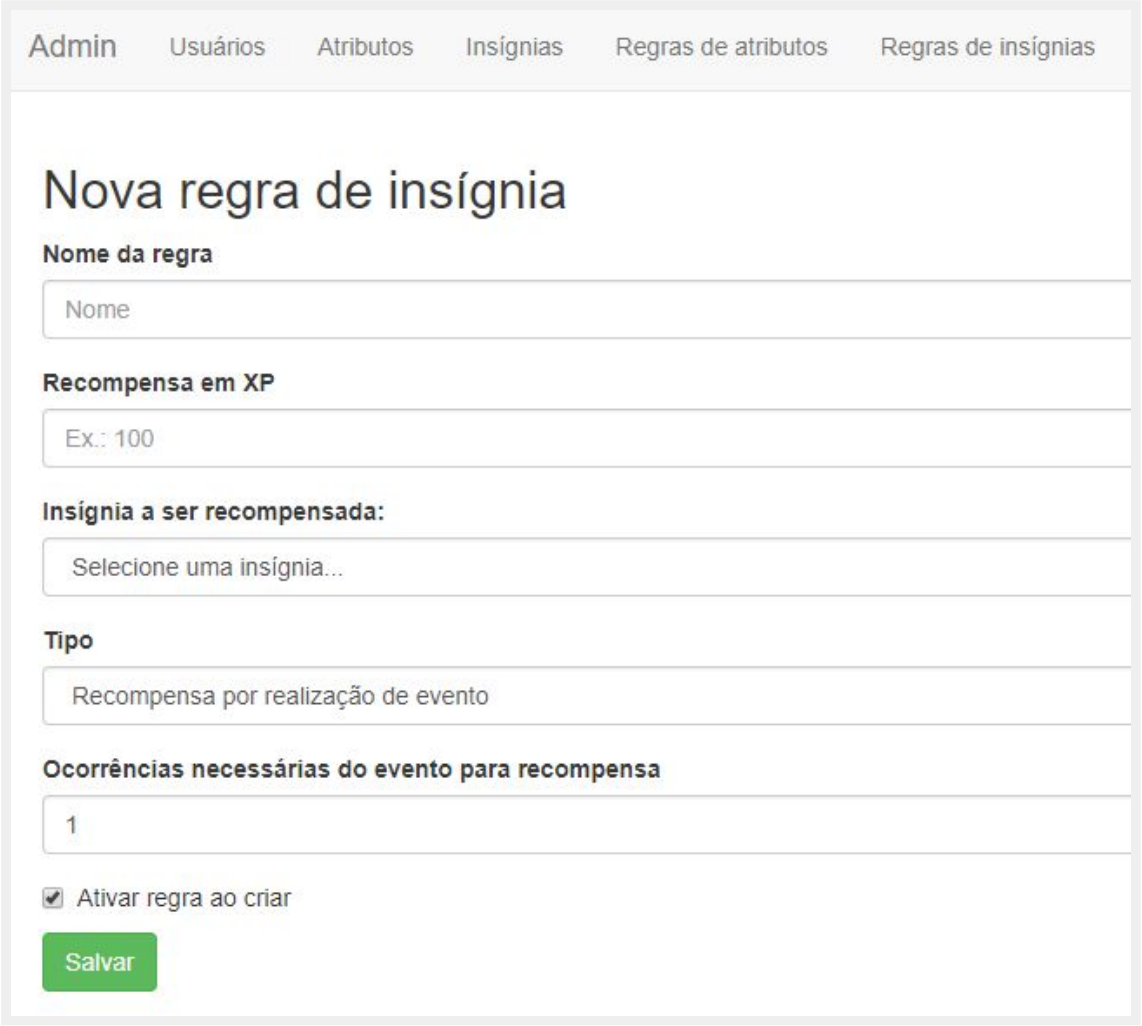
Uma insígnia é uma marca ou selo que registra um acontecimento ou conquista do usuário. Na literatura também pode ser chamada de *badge* ou *achievement*. Neste sistema, existem dois tipos de insígnias: As de evento, e as de atributos.

A partir dos atributos criados no sistema, é possível definir regras para esses atributos, que visam recompensar o usuário por conseguir alcançar determinada marca ou quantia acumulada. Supondo que o atributo ouro seja criado para representar uma moeda dentro do ambiente gamificado, poderia-se, por exemplo, criar regras de insígnia para as marcas de 2000, 5000 e 10000 em quantidade de ouro.

As insígnias de evento recompensam o usuário que realizaram uma ação determinadas vezes. Por exemplo, uma insígnia para aqueles que comentaram no fórum 20 vezes, ou para aqueles que compartilharam conteúdo mais de 10 vezes, etc. A figura 20 mostra o formulário de criação de uma nova regra de insígnia, cujos campos são:

- Nome: nome da regra, serve apenas para identificação visual
- Recompensa em XP: define quantos pontos de experiência o usuário irá receber sempre que realizar a ação.
- Insígnia a ser recompensada: escolha da insígnia previamente criada que deverá ser atribuída ao usuário.
- Tipo: opção com duas escolhas: evento ou acúmulo de atributo. Se a opção for por evento aparece um novo campo para determinar quantas vezes o evento precisa ser realizado, como nas outras regras. Se for por acúmulo de atributo, é necessário informar qual atributo e quantos pontos são necessários acumular naquele atributo para ganhar a insígnia.
- Ativar: se marcado, a regra entra em vigor assim que criada.

Figura 20. Formulário de criação de regras de atributos na interface de administração



Fonte (Autoria própria)

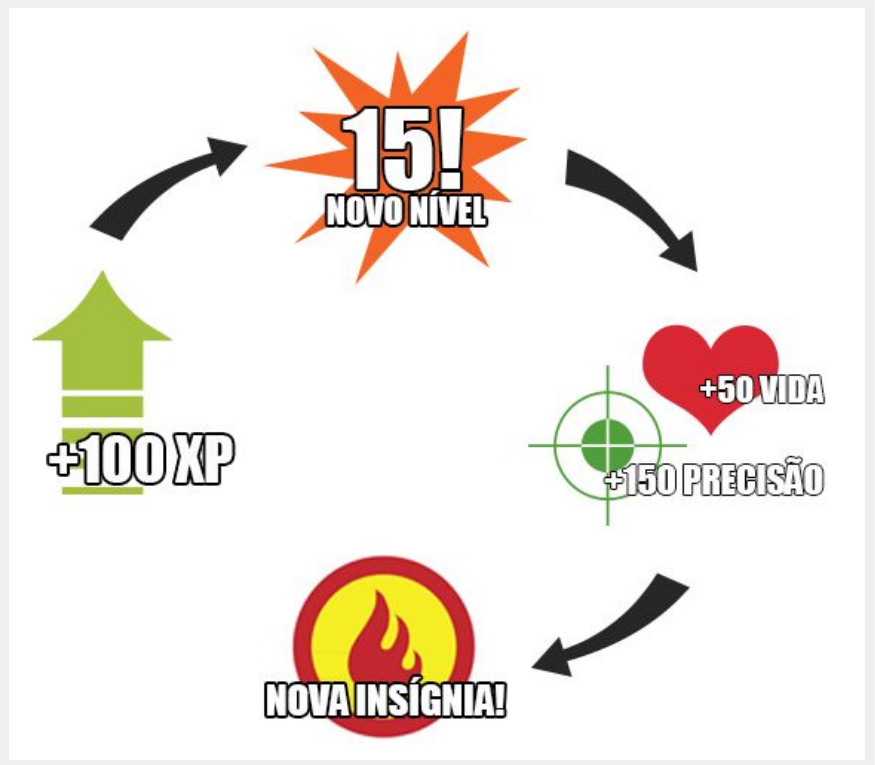
4.3.1.4 Processamento das regras

Uma vez que os elementos de gamificação e as regras associadas a eles tenham sido configurados, a *engine* processará estas regras toda vez que uma nova requisição prevista na integração ocorra, verificando se as pré-condições foram atendidas e atualizando a situação do jogador envolvido. Essa requisição é enviada pelo cliente no momento em que o usuário realizar a ação. Desta forma, a engine irá verificar se o jogador, ao realizar determinada ação, recebeu pontos, ganhou

insígnias, subiu de nível ou ganhou novas posições em determinado *ranking* e irá atualizar estas informações na base de dados.

Durante o processamento de regras é possível que o usuário receba mais de uma recompensa em cascata em função da relação entre os elementos de jogos. Ao receber pontos de experiência, o usuário pode subir de nível. Se alguma regra de recompensa de nível existir, o usuário também irá receber certa quantidade de pontos nos atributos vinculados a essa regra. Por consequência, receber pontos em um determinado atributo pode resultar no ganho de insígnias de atributo, fechando assim o ciclo de recompensas. A figura 21 ilustra, a partir de uma situação hipotética, a dependência entre os elementos de jogos existentes na plataforma.

Figura 21. Ciclo de propagação de recompensas



Fonte (Autoria própria)

4.3.2 API

Toda a comunicação entre a engine e o cliente é feita através de uma API REST, e portanto, via requisições HTTP. A partir da API é possível criar e remover usuários, elementos de gamificação e suas regras associadas; informar a ocorrência de determinado evento; atribuir pontos, níveis, conquistas e demais atributos de gamificação aos usuários, bem como consultar e remover esses dados.

4.3.2.1 Endpoints da API

Dentro do contexto de API's, os *endpoints* nada mais são do que serviços expostos para uso externo. Esses serviços são acessados por requisições HTTP através de uma URL. Em uma API REST, podem surgir *endpoints* com as mesmas URLs. Nesse caso, o que diferencia um do outro é o método HTTP exigido pelo *endpoint*. Os métodos HTTP utilizados na API deste projeto são POST, PUT, DELETE e GET, onde:

- POST: serve para adicionar ou criar um novo recurso
- PUT: serve para atualizar um recurso existente
- DELETE: serve para apagar um recurso
- GET: serve para recuperar ou consultar um recurso

A API conta com *endpoints* para uso externo e para uso interno. Os *endpoints* de uso interno são usados para buscar ou tratar algumas informações pertinentes ao processo de verificação das regras ou criação de conteúdo. Os *endpoints* externos são aqueles que ficam disponíveis para o cliente. A tabela a

seguir, apresenta de maneira sucinta os *endpoints* relevantes da API, por onde é possível gerenciar todo os sistema, caso o painel administrativo não seja suficiente.

Tabela 1. Endpoints da API da plataforma

Método http	**Endpoint**	**Descrição**
GET	/users/	Retorna todos os usuários
GET	/users/{id}	Retorna o usuário com {id} específico
POST	/users/	Insere um novo usuário usando dados passados na requisição
DELETE	/users/	Apaga todos os usuários
DELETE	/users/{id}	Apaga o usuário com {id} específico
PUT	/users/{id}	Atualiza um usuário com {ìd} específico
GET	/attributes/	Retorna todos os atributos
GET	/attributes/{id}	Retorna o atributo com {id} específico
POST	/attributes/	Insere um novo atributo usando dados passados na requisição
DELETE	/attributes/	Apaga todos os atributos
DELETE	/attributes/{id}	Apaga o atributo com {id} específico
PUT	/attributes/{id}	Atualiza um atributo com {ìd} específico
GET	/badges/	Retorna todos os emblemas
GET	/badges/{id}	Retorna o emblema com {id} específico
POST	/badges/	Insere um novo emblema usando dados passados na requisição
DELETE	/badges/	Apaga todos os emblemas

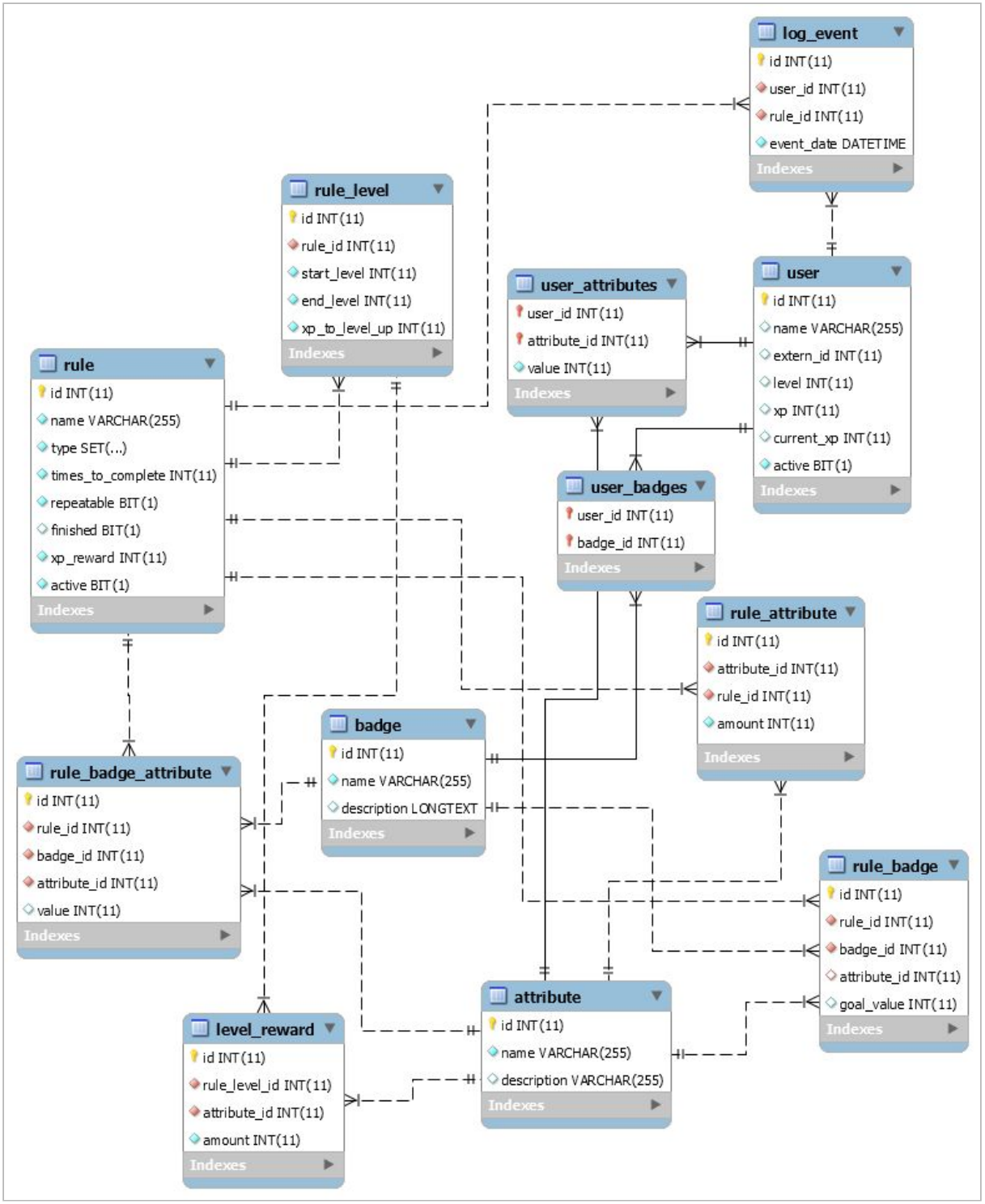
DELETE	/badges/{id}	Apaga o emblema com {id} específico
PUT	/badges/{id}	Atualiza um emblema com {ìd} específico
GET	/rules/	Retorna todas as regras
GET	/rules/{id}	Retorna a regra com {id} específico
POST	/rules/	Insere uma nova regra usando dados passados na requisição
DELETE	/rules/	Apaga todas as regras
DELETE	/rules/{id}	Apaga a regra com {id} específico
PUT	/rules/{id}	Atualiza uma regra com {ìd} específico
POST	/badges/{id}/users/{id}	Atribui uma insígnia a um usuário
DELETE	/badges/{id}/users/{id}	Remove uma insígnia de um usuário
GET	/logs/{logId}	Retorna o log com id específico
GET	/logs/users/{userId}	Retorna todos os logs de determinado usuário
POST	/logs/	Insere um novo log usando dados passados na requisição
DELETE	/logs/rules/{id}/users/{id}	Apaga todos os logs vinculados a uma regra e um usuário
DELETE	/logs/{id}	Apaga o log com {id} específico
DELETE	/logs	Apaga todos os logs do sistema
PUT	/logs/{id}	Atualiza um log com {id} específico
GET	/attributes/rules/{id}	Retorna uma regra de atributo com {id} específico
GET	/attributes/rules	Retorna todas as regras de atributo
DELETE	/attributes/rules/{id}	Remove um regra de atributo
DELETE	/attributes	Remove todas as regras de atributo
POST	/badges/rules	Cria uma nova regra de emblema

PUT	/badges/rules/{id}	Atualiza uma regra de emblema com {id} específico
DELETE	/badges/rules/{id}	Remove uma regra de emblema com {id} específico
GET	/badges/rules/{id}	Retorna uma regra de emblema com {id} específico
POST	/level/rules	Cria uma nova regra de nível
PUT	/level/rules/{id}	Atualiza uma regra de nível com {id} específico
DELETE	/level/rules/{id}	Remove uma regra de nível com {id} específico
DELETE	/level/rules	Apaga todas as regras de nível
GET	/level/rules/{id}	Retorna uma regra de nível com {id} específico

4.3.3 Banco de dados

Os usuário, regras e elementos de gamificação, bem como toda informação relacionada ficam persistidos em um banco de dados estrutural MySQL. Todas essas informações estão disponíveis através dos *endpoints* da API, apresentados no capítulo anterior. A figura 21 mostra o modelo de entidade e relacionamento.

Figura 21. Modelo de entidade e relacionamento do banco de dados



Fonte (Autoria própria, feita com ferramenta MySQL Workbench)

5. TECNOLOGIAS UTILIZADAS

Das tecnologias utilizadas no desenvolvimento projeto do projeto, desde o planejamento até o estágio de testes, destacam-se, separando por categorias ou componente, as seguintes tecnologias:

- Engine e API
    - Java 8
    - Spring boot framework
- Banco de dados
    - MySQL
    - HeidiSQL (IDE)
    - MySQL Workbench (modelagem)
- Interface de administração
    - HTML
    - CSS
    - Javascript
    - AngularJS
- Serviços
    - Java 8
    - JPA
    - Hibernate

6. CONSIDERAÇÕES FINAIS

Gamificação é uma técnica que ainda está em fase de crescimento e ainda está sendo descoberta por empresas e instituições que buscam de alguma maneira motivar seu público alvo. Estudar a fundo o conceito de gamificação e as aplicações que usam e oferecem serviços relacionados a essa prática, resultou no desenvolvimento de uma plataforma alternativa, diferente e com diversas portas abertas para crescer. A partir das tecnologias escolhidas, foi possível estabelecer uma camada de integração abrangente o suficiente para abrigar o desenvolvimento de ambientes de gamificação para *mobile*, aplicações web e *desktop*.

Em relação ao processo de desenvolvimento, um dos principais desafios foi arquitetar a estrutura. Era necessário uma ferramenta que conversasse com a aplicação do cliente e que de algum modo houvesse uma conexão estabelecida entre as duas partes. A primeira ideia era centralizar a engine e todos os componentes do projeto em um servidor que se comunicaria através de autenticação com todos os clientes via API. Essa ideia foi descartada devido às possíveis complicações de performance e segurança, que aumentariam demais a complexidade do trabalho. Como solução, decidiu-se que cada cliente teria a sua própria versão completa da plataforma, instalada em seu servidor de preferência, onde a integração continuaria sendo via API. Assim, o cliente teria domínio sobre os dados e sobre a disponibilidade da plataforma.

Além disso, durante o planejamento e desenvolvimento do projeto, algumas ideias tiveram que ser descartadas em detrimento ao escopo do trabalho devido a sua natural complexidade. As missões, um elemento de jogo importante na área de

gamificação, foi uma delas. Nos jogos, uma missão é uma sequência de passos ou etapas que levam a um objetivo maior. Essas etapas podem ser ordenadas, repetitivas, cíclicas e necessitar de condições temporais. Essas características exigem uma estrutura maior e um processamento de regras mais inteligente, pois divergem muito da maneira como os outros elementos são processados.

Em relação aos trabalhos futuros, além de introduzir as missões como um quinto elemento de jogo, diversas outras pequenas e grandes melhorias podem ser feitas, como:

- Permitir a adição de imagem às insígnias: Atualmente o cliente precisa processar isso separadamente em sua aplicação.
- Adicionar informativos nos campos da interface de administração explicando o que cada um significa.
- Tratamento de erros da API e da interface de administração: Em ambos os casos, os possíveis erros estão sendo tratados de maneira genérica.
- Permissões: Atualmente não existe tratamento de permissões nas funcionalidades da API e na interface de administração. É necessário a criação de um sistema de autenticação de usuário.

Além dos problemas mencionados, que se enquadram em melhorias, uma adição interessante seria uma interface de análise e estatísticas das atividades dos usuários. Essa funcionalidade serviria para a administração ter um *feedback* de como o seu programa de gamificação está indo e fazer os devidos ajustes à medida que for necessário. Para isso, além da interface, seria necessário criar uma estrutura de *logs* adicional, que envolve a criação de novas tabelas do banco,

alterações no processamento das regras para registrar esses *logs* e serviços para gerenciamento na API.

Por fim, ainda que a instalação da plataforma não seja tão trivial, o seu uso e administração requer muito pouco conhecimento técnico. Esta condição só foi possível graças à interface de administração que foi desenvolvida para dar suporte ao gerenciamento e a maneira simples de integração entre o cliente e servidor. Além disso, a plataforma também se mostra flexível quanto à forma de exibição das notificações das conquistas e atualizações dos jogadores, oferecendo uma estrutura de *feedback* que permite ao desenvolvedor da aplicação cliente escolher a melhor forma de tratar e apresentar estes dados.

REFERÊNCIAS

VENTUREBEAT. **Pwc: game industry to grow nearly 5% annually through 2020**. Disponível em: http://venturebeat.com/2016/06/08/the-u-s-and-global-game-industries-will-grow-a- healthy-amount-by-2020-pwc-forecasts/>. Acesso em: 29 jun. 2017.

VENTUREBEAT. **Video games will become a $99.6b industry this year as mobile overtakes consoles and pcs**. Disponível em: <https://venturebeat.com/2016/06/08/the-u-s-and-global-game-industries-will-grow-a-healthy-amount-by-2020-pwc-forecasts/>. Acesso em: 29 jun. 2017.

LI, Wei; GROSSMAN, Tovi; FITZMAURICE, George. **Gamified Tutorial System For First Time AutoCAD Users**. UIST ’12, October 7–10, 2012, Cambridge, Massachusetts, USA.

VIANNA, Ysmar; VIANNA, Maurício; MEDINA, Bruno; TANAKA,Samara. **Gamification, Inc.: Como reinventar empresas a partir de jogos**. MJV Press: Rio de Janeiro, 2013.

ZICHERMANN, Gabe; CUNNINGHAM, Christopher. Gamification by Design: **Implementing Game Mechanics in Web and Mobile Apps**. Sebastopol, CA: O’Reilly Media, Inc. 2011.

BUSARELLO, Raul Inácio. **Gamification: princípios e estratégias**. Pimenta cultural, são paulo, p. 126, 2016.

ROBSON, K. et al. **Game on: Engaging customers and employees through gamification**. Business Horizons, Kelley School of Business, Indiana University, v. 59, n. 1, p. 29-36, jan./fev. 2016.

YOUTUBE. **Bottle bank arcade - thefuntheory.com - rolighetsteorin.se**. Disponível em: <https://www.youtube.com/watch?v=zsihjmu-muo>. Acesso em: 29 jun. 2017.

YOUTUBE. **Happy atoms - digital and physical chemistry set - indiegogo campaign**. Disponível em: <https://www.youtube.com/wat ch?v=w0djyddtqyo>. Acesso em: 29 jun. 2017.

MCGONIGAL, Jane. **A realidade em jogo: porque os games nos tornam melhores e como eles podem mudar o mundo**. Rio de Janeiro: Best Seller, 2012.

KIM, Bohyun. **Understanding gamification**. 2 ed. Chicago, USA: American Library Association, 2015.

MATIAS, André Victória. **Aplicativo para Uso de Ludificação no Tratamento de Diabetes Mellitus Tipo 1**. 2015. Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, Santa Catarina, 2015.

KIRRIEMUIR, J.; MCFARLANE, A. (2004) **"Literature Review in Games and Learning"**. FutureLab. Disponível em<http://archive.futurelab.org.uk/resources/documents/lit_reviews/Games_Review.pdf>

WILLIAMSON, B. (2009) **"Computer games, schools, and young people : A report for educators on using games for learning"**. FutureLab. Disponível em: <http://archive.futurelab.org.uk/resources/documents/project_reports/becta/Games_and_Learning_educators_report.pdf>

RABIN, Steve. **Introduction to game development**. 2 ed. Boston, USA: Cengage Learning, 2010. 58 p.

PEWINTERNET. **Mobile fact sheet**. Disponível em: <http://www.pewinternet.org/fact-sheet/mobile/>. Acesso em: 29 jun. 2017.

DETERDING, S. et al. From game design elements to gamefulness: defining "gamification". **Envisioning Future Media Environments**, em: the 15th International Academic MindTrek Conference, Tampere, Finland, p. 9-15, set. 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=2181040>. Acesso em: 29 jun. 2017.

GEOAWESOMENESS. **Location-based marketing – foursquare analysis**. Disponível em: <http://geoawesomeness.com/knowledge-base/location-based-marketing/location-based-marketing-foursquare-analysis/>. Acesso em: 30 jun. 2017.

GITHUB. **Gamification-engine**. Disponível em: <https://github.com/actidoo/gamification-engine>. Acesso em: 30 jun. 2017.

Robin Hunicke, Marc Leblanc, and Robert Zubek, "**MDA: A Formal Approach to Game Design and Game Research**," in Proceedings of the Challenges in Games AIWorkshop, Nineteenth National Conference of Artificial Intelligence (San Jose, CA: AAAI Press, 2004), 2.

Markets and Markets. (2016). **Gamification Market by Solution (Consumer driven and Enterprise driven), Applications (Sales and Marketing), Deployment Type (On-Premises and Cloud), User Type (Large Enterprise, SMBs), Industry and Region - Global Forecast to 2020**. Disponível em: <https://www.marketsandmarkets.com/Market-Reports/gamification-market-991.html>. Acesso em: 24 de maio, 2018.

Seaborn, K., & Fels, D. I. (2015). **Gamification in theory and action: A survey**. International Journal of Human-Computer Studies, 74(2), 14Ð31. Disponível em: <http://doi.org/10.1016/j.ijhcs.2014.09.006>

BRANDUNIC. **Gamification for Marketers: a Beginner's Guide**. Disponível em: <http://branduniq.com/2013/gamification-for-marketers/>. Acesso em: 01 jun. 2018.

HABITICA. **Página inicial do habitica**. Disponível em: <https://habitica.com/>. Acesso em: 01 jun. 2018

TCHCRUNCH. **Happy Atoms launches to teach kids about the wonders of molecules**. Disponível em: <https://techcrunch.com/2016/06/28/happy-atoms-launches-to-teach-kids-about-the-wonders-of-molecules/> Acesso em: 01 jun. 2018

ENGADGET. **Nike+ Running iOS update adds social challenges, trash talk.** Disponível em: <https://www.engadget.com/2013/07/24/nike-plus-running-ios-app-update/>. Acesso em: 01 jun. 2018

DUOLINGO. **Site da aplicação**. Disponível em: <https://www.duolingo.com/>. Acesso em: 01 jun. 2018

GETBADGES. **Site da plataforma**. Disponível em: <https://getbadges.io/>. Acesso em: 01 jun. 2018

GETBADGES. **Site da plataforma**. Disponível em: <https://badgeville.com/>. Acesso em: 01 jun. 2018

Apêndice A - Artigo do trabalho

Uma Plataforma Para Integração e Processamento de Regras e Elementos de Gamificação em Sistemas

Thiago de Campos

Departamento de Informática e estatística
Universidade Federal de Santa Catarina (UFSC)

thiagocamposde@gmail.com

Abstract. *Gamification is a practice that aims to engage and motivate people in tasks and day-to-day contexts through game features and elements. Companies are using this technique to attract and retain customers, improve employee performance, gain more success in training and teaching methods, and more. However, the development of an efficient and long-lasting gamification strategy, which arouses people's intrinsic motivation, requires a series of analyzes and studies that end up being the great challenge for those who want to obtain positive results. The purpose of this work is to create an easy integration tool that facilitates the development of gamic environments. For this, a manageable platform was developed that combines the creation of game elements, such as experience points, levels, badges, attributes, with the creation and processing of rules that aim to automate the assignment of these elements to system users. With an administrative dashboard and the a simple feedback system, the developed solution can serve as an alternative to other market platforms.*

Keywords: *gamification, gamification platform , game elements, games, engine, motivation, engagement.*

Resumo. *Gamificação é uma prática que tem por objetivo engajar e motivar pessoas em tarefas e contextos do dia-a-dia através de características e elementos de jogos. Empresas estão usando esta técnica para atrair e manter clientes, melhorar o desempenho de funcionários e colaboradores, obter mais sucesso em treinamentos e métodos de ensino,*

entre outros. Todavia, a elaboração de uma estratégia de gamificação eficiente e duradoura, que desperte a motivação intrínseca das pessoas, requer uma série de análise e estudos que acabam sendo o grande desafio para quem deseja obter resultados positivos. A proposta deste trabalho é criar uma ferramenta de fácil integração, que facilite o desenvolvimento de ambientes gamificados. Para isso, foi desenvolvida uma plataforma gerenciável, unindo a criação de elementos de jogos, como pontos de experiência, níveis, insígnias, atributos, com a criação e processamento de regras que visam automatizar a atribuição destes elementos aos usuários do sistema. Com um painel administrativo e um sistema de feedback simples, a solução desenvolvida pode servir como uma alternativa a outras plataformas do mercado.

Palavras-chave: *gamificação, plataforma de gamificação, elementos de jogos, jogos, engine, motivação, engajamento*

1. Introdução

Com a invenção do computador e posteriormente o surgimento dos jogos eletrônicos, as possibilidades acerca da criação e desenvolvimento de jogos aumentaram drasticamente, dando a esse mercado um crescimento e relevância que perduram até hoje (Kim, 2015). O grande aumento da aquisição de *smartphones* e computadores pessoais nos últimos anos, junto com a expansão da internet, vem fazendo com que as pessoas estejam cada vez mais conectadas e acostumadas a interagirem através deste meio (KIM, 2015). Unindo esta nova maneira de viver e interagir da sociedade com a capacidade dos jogos de proporcionar fortes experiências e engajar pessoas, novas abordagens em torno deste mercado acabaram surgindo, como é o caso de gamificação (do inglês, *gamification*).

Gamificação é uma técnica com o objetivo de extrair os elementos positivos dos jogos, tais como pontuação, conquistas, objetivos, regras, mecânicas, dinâmicas, sistema de *feedback*, etc., e inseri-los em outros tipos de atividades e contextos no intuito de torná-los mais atrativos, e por consequência, provocar uma mudança de comportamento nas pessoas envolvidas, tornando-as mais engajadas, motivadas e satisfeitas (ZICHERMANN e CUNNINGHAM, 2011; Kim, 2015).

Apesar do conceito ser simples, a incorporação de gamificação em um determinado contexto não é uma tarefa trivial. Para alcançar os resultados desejados, ou seja, envolver as

pessoas em uma experiência lúdica e fazer com que engajem na atividade proposta, é preciso, além de traçar uma boa estratégia de execução, planejar com muito cuidado os diversos aspectos que compõem uma boa estratégia de gamificação (KIETZMANN et al., 2016).

Diante de tantos fatores a serem considerados e planejados, é natural que em muitos casos, a dificuldade em se implementar uma boa estratégia de gamificação em um sistema se encontra mais na concepção da ideia do que no desenvolvimento do software propriamente dito. A finalidade deste trabalho, seguindo essa linha de raciocínio, é amenizar a parte técnica, propondo como solução uma plataforma que atuará como um facilitador e ferramenta de auxílio para a implantação de gamificação em sistemas, tirando parte do peso da implementação de sistemas gamificados, servindo também como um guia de gamificação através das funcionalidades disponíveis e do fluxo com que o sistema é configurado.

2. Desenvolvimento do trabalho

Este trabalho tem como proposta o desenvolvimento de uma plataforma que facilite a introdução de gamificação em sistemas *web*, *desktop* e *mobile* com o objetivo de tirar das mãos do criador a responsabilidade e o esforço de desenvolver a solução técnica, abrindo margem para elaboração do plano estratégico e diminuindo o tempo e custo total do projeto.

Levando em consideração os estudos realizados em torno das outras soluções de mercado e dos objetivos deste trabalho, foi necessário criar uma solução multiplataforma, de fácil configuração, integração e entendimento, que fosse também escalável e modelada de forma que facilitasse a criação de novos recursos no futuro. Tendo estes requisitos e o conceito de gamificação em vista, a primeira tarefa foi definir quais mecânicas de jogos seriam suportadas pela plataforma e como elas seriam atribuídas aos usuários finais, o que resultou na criação de dois conceitos chaves pelo qual toda a plataforma se sustenta: **os elementos de gamificação** e as **regras de gamificação**.

2.1 Elementos de gamificação

Para este trabalho, foram adotados cinco elementos de gamificação, os quais representam uma ou mais mecânicas de jogos:

- Pontos de experiência
- Nível

- Insígnias
- Atributos
- Rankings

Com exceção do elemento atributo, que é apenas uma representação abstrata de qualquer mecânica numérica (ex.: pontos de vida, ouro, fama, reputação, etc.), todos os outros representam uma das mecânicas apresentadas na seção 2.5.1, faltando apenas as missões e customização, que não entram no escopo.

2.2 Regras de gamificação

Para que a plataforma pudesse funcionar de maneira automática após sua configuração, foi necessária a criação do conceito de regras de gamificação. Essas regras nada mais são do que pré-condições atreladas ou não a ações que precisam ser cumpridas em determinado contexto para que os elementos de gamificação anteriormente mencionados sejam atribuídos aos usuários finais. Exemplos hipotéticos:

- ”Toda vez que um usuário compartilhar uma notícia, ele receberá 200 pontos de experiência”;
- “Ao fazer 5 comentários, o usuário ganhará 100 pontos de atributo fama”;
- “Do nível 1 ao 10, sempre que atingir 1000 pontos de experiência, o usuário subirá de nível”;
- “Ao acumular 20000 pontos no atributo fama, o usuário receberá a insígnia *um astro do rock*”.

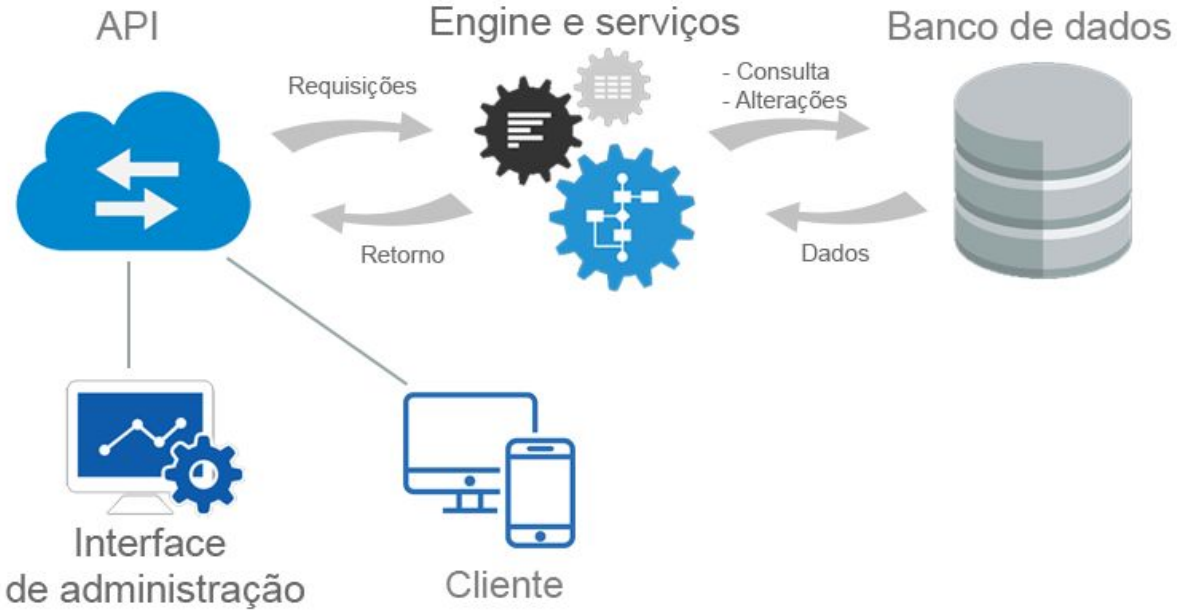
Em outras palavras as regras servem para definir quando e como que os usuários devem ganhar pontos de experiência, pontos de atributos, subir de nível, conquistar uma insígnia, etc. Para cada elemento de gamificação, foi elaborado um tipo de regra específica, cuja criação e configuração ficam a cargo do responsável pelo ambiente de gamificação.

2.3 Componentes da plataforma

Uma vez definidos os conceitos principais, foi necessário planejar uma estrutura para criar, gerenciar, processar e armazenar os elementos e regras de gamificação mencionados anteriormente. Para alcançar esses objetivos, foi definido que a plataforma seria composta de cinco componentes principais: (1) uma engine, que é responsável pelo processamento e

validação das regras de gamificação; (2) uma central de serviços, que é por onde todos os recursos da plataforma são criados e gerenciados; (3) uma API, que serve de via de comunicação entre o sistema-cliente, a engine, e os serviços; (4) uma estrutura de banco de dados, onde os elementos, as regras e os usuários que farão parte do ambiente de gamificação serão armazenados; e (5), uma interface de administração da plataforma. A figura 1 mostra uma representação dos cinco componentes e suas respectivas interações.

Figura 1. Arquitetura e representação dos componentes da plataforma



Fonte (Autoria própria)

2.4 Definição do funcionamento da plataforma

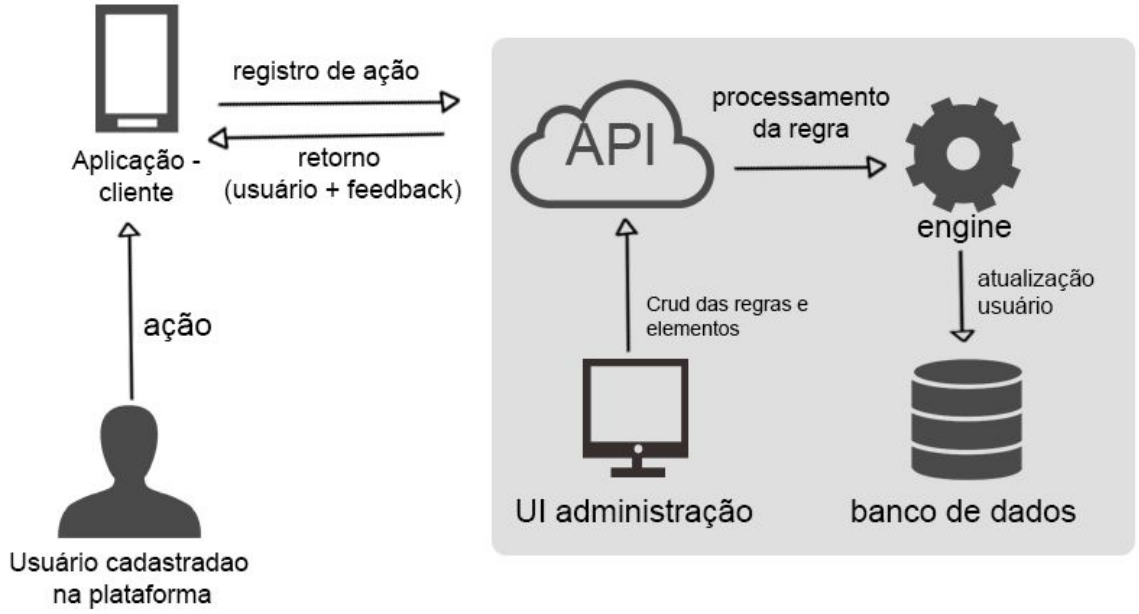
O primeiro passo é baixar, instalar e configurar a plataforma em seu local de jogos e regras de gamificação desejáveis, o que pode ser feito via interface de administração ou API, o desenvolvedor do ambiente gamificado precisa popular o banco de dados da plataforma com os usuários que farão parte do sistema.

Após esta etapa, o desenvolvedor precisa acessar o código fonte do seu sistema, seja ele um site, um app, ou outro programa e fazer a integração com a plataforma. Essa integração se dá por meio de registro de atividades dos usuários, que são feitos via requisições HTTP para a API da plataforma, através do seguinte *endpoint*:

http://endereçodoservidor/api/event-rule/{idUsuario}/{idRule}, onde, o {idUsuario} é o identificador do usuário na plataforma e o {idRule} é o identificador da regra que deve ser processada, que nesse caso também representa um evento ou atividade realizada pelo usuário. Dessa forma, por exemplo, se foi criada uma regra para recompensar usuários que fazem *login* diariamente, é necessário colocar essa chamada de requisição na parte do código onde é efetuado o *login*, especificando corretamente na chamada qual a regra e qual o usuário que efetuou *login*.

Ao receber uma requisição do cliente, com informações do usuário e a regra em questão, a engine irá registrar a atividade no banco de dados e fará o processamento de dados para verificar, através das pré-condições, se a regra foi concluída e atribuir as devidas recompensas ao usuário, retornando para o cliente um *feedback* contento o usuário atualizado e tudo que foi atribuído a ele a partir da ação. Com essas informações, o desenvolvedor programar seu sistema para dar o *feedback* adequado ao usuário que está utilizando sua aplicação, seja através de notificações, *popups*, mensagens, atualizações gráficas, etc. A figura 2 mostra de maneira sucinta toda a interação, desde a ação do usuário até o processamento completo da regra.

Figura 2. Representação do fluxo de processamento de regras da plataforma

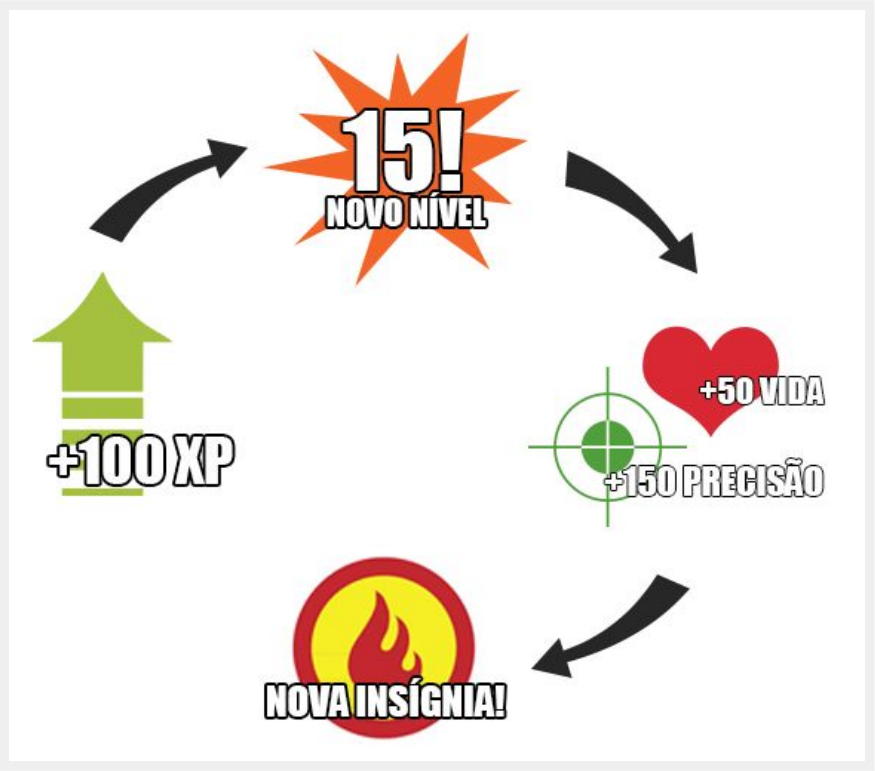


Fonte: (Autoria própria)

2.5 Processamento das regras

Durante o processamento de regras é possível que o usuário receba mais de uma recompensa em cascata em função da relação entre os elementos de jogos. Ao receber pontos de experiência, o usuário pode subir de nível. Se alguma regra de recompensa de nível existir, o usuário também irá receber certa quantidade de pontos nos atributos vinculados a essa regra. Por consequência, receber pontos em um determinado atributo pode resultar no ganho de insígnias de atributo, fechando assim o ciclo de recompensas. A figura 21 ilustra, a partir de uma situação hipotética, a dependência entre os elementos de jogos existentes na plataforma.

Figura 21. Ciclo de propagação de recompensas



Fonte (Autoria própria)

3. Considerações finais

Gamificação é uma técnica que ainda está em fase de crescimento e ainda está sendo descoberta por empresas e instituições que buscam de alguma maneira motivar seu público alvo. Estudar a fundo o conceito de gamificação e as aplicações que usam e oferecem serviços relacionados a essa prática, resultou no desenvolvimento de uma plataforma alternativa, diferente e com diversas portas abertas para crescer. A partir das tecnologias escolhidas, foi possível estabelecer uma camada de integração abrangente o suficiente para abrigar o desenvolvimento de ambientes de gamificação para *mobile*, aplicações web e *desktop*.

Em relação ao processo de desenvolvimento, um dos principais desafios foi arquitetar a estrutura. Era necessário uma ferramenta que conversasse com a aplicação do cliente e que de algum modo houvesse uma conexão estabelecida entre as duas partes. A primeira ideia era centralizar a engine e todos os componentes do projeto em um servidor que se comunicaria através de autenticação com todos os clientes via API. Essa ideia foi descartada devido às

possíveis complicações de performance e segurança, que aumentariam demais a complexidade do trabalho. Como solução, decidiu-se que cada cliente teria a sua própria versão completa da plataforma, instalada em seu servidor de preferência, onde a integração continuaria sendo via API. Assim, o cliente teria domínio sobre os dados e sobre a disponibilidade da plataforma.

Além disso, durante o planejamento e desenvolvimento do projeto, algumas ideias tiveram que ser descartadas em detrimento ao escopo do trabalho devido a sua natural complexidade. As missões, um elemento de jogo importante na área de gamificação, foi uma delas. Nos jogos, uma missão é uma sequência de passos ou etapas que levam a um objetivo maior. Essas etapas podem ser ordenadas, repetitivas, cíclicas e necessitar de condições temporais. Essas características exigem uma estrutura maior e um processamento de regras mais inteligente, pois divergem muito da maneira como os outros elementos são processados.

Em relação aos trabalhos futuros, além de introduzir as missões como um quinto elemento de jogo, diversas outras pequenas e grandes melhorias podem ser feitas, como:

- Permitir a adição de imagem às insígnias: Atualmente o cliente precisa processar isso separadamente em sua aplicação.
- Adicionar informativos nos campos da interface de administração explicando o que cada um significa.
- Tratamento de erros da API e da interface de administração: Em ambos os casos, os possíveis erros estão sendo tratados de maneira genérica.
- Permissões: Atualmente não existe tratamento de permissões nas funcionalidades da API e na interface de administração. É necessário a criação de um sistema de autenticação de usuário.

Além dos problemas mencionados, que se enquadram em melhorias, uma adição interessante seria uma interface de análise e estatísticas das atividades dos usuários. Essa funcionalidade serviria para a administração ter um *feedback* de como o seu programa de gamificação está indo e fazer os devidos ajustes à medida que for necessário. Para isso, além da interface, seria necessário criar uma estrutura de *logs* adicional, que envolve a criação de novas tabelas do banco, alterações no processamento das regras para registrar esses *logs* e serviços para gerenciamento na API.

Por fim, ainda que a instalação da plataforma não seja tão trivial, o seu uso e administração requer muito pouco conhecimento técnico. Esta condição só foi possível graças à interface de administração que foi desenvolvida para dar suporte ao gerenciamento e a maneira simples de integração entre o cliente e servidor. Além disso, a plataforma também se mostra flexível quanto à forma de exibição das notificações das conquistas e atualizações dos jogadores, oferecendo uma estrutura de *feedback* que permite ao desenvolvedor da aplicação cliente escolher a melhor forma de tratar e apresentar estes dados.

REFERÊNCIAS

VENTUREBEAT. **Pwc: game industry to grow nearly 5% annually through 2020**. Disponível em: http://venturebeat.com/2016/06/08/the-u-s-and-global-game-industries-will-grow-a- healthy-amount-by-2020-pwc-forecasts/>. Acesso em: 29 jun. 2017.

VENTUREBEAT. **Video games will become a $99.6b industry this year as mobile overtakes consoles and pcs**. Disponível em: <https://venturebeat.com/2016/06/08/the-u-s-and-global-game-industries-will-grow-a-healthy-amount-by-2020-pwc-forecasts/>. Acesso em: 29 jun. 2017.

LI, Wei; GROSSMAN, Tovi; FITZMAURICE, George. **Gamified Tutorial System For First Time AutoCAD Users**. UIST ’12, October 7–10, 2012, Cambridge, Massachusetts, USA.

VIANNA, Ysmar; VIANNA, Maurício; MEDINA, Bruno; TANAKA,Samara. **Gamification, Inc.: Como reinventar empresas a partir de jogos**. MJV Press: Rio de Janeiro, 2013.

ZICHERMANN, Gabe; CUNNINGHAM, Christopher. Gamification by Design: **Implementing Game Mechanics in Web and Mobile Apps**. Sebastopol, CA: O’Reilly Media, Inc. 2011.

BUSARELLO, Raul Inácio. **Gamification: princípios e estratégias**. Pimenta cultural, são paulo, p. 126, 2016.

ROBSON, K. et al. **Game on: Engaging customers and employees through gamification**. Business Horizons, Kelley School of Business, Indiana University, v. 59, n. 1, p. 29-36, jan./fev. 2016.

YOUTUBE. **Bottle bank arcade - thefuntheory.com - rolighetsteorin.se**. Disponível em: <https://www.youtube.com/watch?v=zsihjmu-muo>. Acesso em: 29 jun. 2017.

YOUTUBE. **Happy atoms - digital and physical chemistry set - indiegogo campaign**. Disponível em: <https://www.youtube.com/wat ch?v=w0djyddtqyo>. Acesso em: 29 jun. 2017.

MCGONIGAL, Jane. **A realidade em jogo: porque os games nos tornam melhores e como eles podem mudar o mundo**. Rio de Janeiro: Best Seller, 2012.

KIM, Bohyun. **Understanding gamification**. 2 ed. Chicago, USA: American Library Association, 2015.

MATIAS, André Victória. **Aplicativo para Uso de Ludificação no Tratamento de Diabetes Mellitus Tipo 1**. 2015. Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, Santa Catarina, 2015.

KIRRIEMUIR, J.; MCFARLANE, A. (2004) **"Literature Review in Games and Learning"**. FutureLab. Disponível em<http://archive.futurelab.org.uk/resources/ documents/lit_reviews/Games_Review.pdf>

WILLIAMSON, B. (2009) **"Computer games, schools, and young people : A report for educators on using games for learning"**. FutureLab. Disponível em: <http://archive.futurelab.org.uk/resources/documents/project_reports/becta/Games_and_Learning_educators_report.pdf>

RABIN, Steve. **Introduction to game development**. 2 ed. Boston, USA: Cengage Learning, 2010. 58 p.

PEWINTERNET. **Mobile fact sheet**. Disponível em: <http://www.pewinternet .org/fact-sheet/mobile/>. Acesso em: 29 jun. 2017.

DETERDING, S. et al. From game design elements to gamefulness: defining "gamification". **Envisioning Future Media Environments**, em: the 15th International Academic MindTrek Conference, Tampere, Finland, p. 9-15, set. 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=2181040>. Acesso em: 29 jun. 2017.

GEOAWESOMENESS. **Location-based marketing – foursquare analysis**. Disponível em: <http://geoawesomeness.com/knowledge-base/location-based-marketing/location-based-marketing-foursquare-analysis/>. Acesso em: 30 jun. 2017.

GITHUB. **Gamification-engine**. Disponível em: <https://github.com/actidoo/gamification-engine>. Acesso em: 30 jun. 2017.

Robin Hunicke, Marc Leblanc, and Robert Zubek, "**MDA: A Formal Approach to Game Design and Game Research**," in Proceedings of the Challenges in Games AIWorkshop, Nineteenth National Conference of Artificial Intelligence (San Jose, CA: AAAI Press, 2004), 2.

Markets and Markets. (2016). **Gamification Market by Solution (Consumer driven and Enterprise driven), Applications (Sales and Marketing), Deployment Type (On-Premises and Cloud), User Type (Large Enterprise, SMBs), Industry and Region - Global Forecast to 2020**. Disponível em: <https://www.marketsandmarkets.com/Market-Reports/gamification-market-991.html>. Acesso em: 24 de maio, 2018.

Seaborn, K., & Fels, D. I. (2015). **Gamification in theory and action: A survey**. International Journal of Human-Computer Studies, 74(2), 14Ð31. Disponível em: <http://doi.org/10.1016/j.ijhcs.2014.09.006>

BRANDUNIC. **Gamification for Marketers: a Beginner's Guide**. Disponível em: <http://branduniq.com/2013/gamification-for-marketers/>. Acesso em: 01 jun. 2018.

HABITICA. **Página inicial do habitica**. Disponível em: <https://habitica.com/>. Acesso em: 01 jun. 2018

TCHCRUNCH. **Happy Atoms launches to teach kids about the wonders of molecules**. Disponível em: <https://techcrunch.com/2016/06/28/happy-atoms-launches-to-teach-kids-about-the-wonders-of-molecules/> Acesso em: 01 jun. 2018

ENGADGET. **Nike+ Running iOS update adds social challenges, trash talk.** Disponível em: <https://www.engadget.com/2013/07/24/nike-plus-running-ios-app-update/>. Acesso em: 01 jun. 2018

DUOLINGO. **Site da aplicação**. Disponível em: <https://www.duolingo.com/>. Acesso em: 01 jun. 2018

GETBADGES. **Site da plataforma**. Disponível em: <https://getbadges.io/>. Acesso em: 01 jun. 2018

GETBADGES. **Site da plataforma**. Disponível em: <https://badgeville.com/>. Acesso em: 01 jun. 2018

APÊNDICE B - CÓDIGO FONTE

O código fonte completo pode ser encontrado também no repositório GitHub, em:

<https://github.com/thiagocamposde/gamification-engine>

ApiGamifyEngine.java

```java
package br.ufsc.tcc.gamifyEngine;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 * Hello world!
 *
 */

@Configuration
@EnableAutoConfiguration
@ComponentScan("br.ufsc.tcc.gamifyEngine")
@SpringBootApplication(scanBasePackages = {
    "br.ufsc.tcc.gamifyEngine"
})
public class ApiGamifyEngine {
    public static void main(String[] args) {
        SpringApplication.run(ApiGamifyEngine.class, args);
    }
}
```

LevelRewardKey.java

```java
package br.ufsc.tcc.gamifyEngine.compositeKeys;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class LevelRewardKey implements Serializable {

	private static final long serialVersionUID = 1L;

	@Column(name = "attribute_id", nullable = false)
	private int attribute;

	@Column(name = "rule_level_id", nullable = false)
	private int ruleLevel;

	public LevelRewardKey() {
		// TODO Auto-generated constructor stub
	}

	public LevelRewardKey(int attribute, int ruleLevel) {
		this.attribute = attribute;
		this.ruleLevel = ruleLevel;
	}

	public int getAttribute() {
		return attribute;
	}

	public void setAttribute(int attribute) {
		this.attribute = attribute;
	}

	public int getRuleLevel() {
```

```
		return ruleLevel;
	}

	public void setRuleLevel(int ruleLevel) {
		this.ruleLevel = ruleLevel;
	}

}
```

RuleAttributeKey.java

```
package br.ufsc.tcc.gamifyEngine.compositeKeys;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class RuleAttributeKey implements Serializable {

	private static final long serialVersionUID = 1L;

	@Column(name = "user_id", nullable = false)
	private int user;

	@Column(name = "attribute_id", nullable = false)
	private int attribute;

	public RuleAttributeKey() {
		// TODO Auto-generated constructor stub
	}

	public RuleAttributeKey(int user, int attribute) {
		this.user = user;
		this.attribute = attribute;
	}

	public int getUser() {
```

```
            return user;
      }

      public void setUser(int user) {
            this.user = user;
      }

      public int getAttribute() {
            return attribute;
      }

      public void setAttribute(int attribute) {
            this.attribute = attribute;
      }
}
```

AdminController.java

```
package br.ufsc.tcc.gamifyEngine.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class AdminController {
       @RequestMapping(value="/admin",method = RequestMethod.GET)
      public String adminHomePage(){
          return "index";
      }
}
```

RestApiController.java

```
package br.ufsc.tcc.gamifyEngine.controller;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import br.ufsc.tcc.gamifyEngine.compositeKeys.LevelRewardKey;
import br.ufsc.tcc.gamifyEngine.compositeKeys.RuleAttributeKey;
import br.ufsc.tcc.gamifyEngine.model.Attribute;
import br.ufsc.tcc.gamifyEngine.model.FeedbackAttributes;
import br.ufsc.tcc.gamifyEngine.model.FeedbackXp;
import br.ufsc.tcc.gamifyEngine.model.Badge;
import br.ufsc.tcc.gamifyEngine.model.Engine;
import br.ufsc.tcc.gamifyEngine.model.LevelReward;
import br.ufsc.tcc.gamifyEngine.model.LogEvent;
import br.ufsc.tcc.gamifyEngine.model.Rule;
import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleBadge;
import br.ufsc.tcc.gamifyEngine.model.RuleBadgeAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleLevel;
import br.ufsc.tcc.gamifyEngine.model.User;
import br.ufsc.tcc.gamifyEngine.model.UserAttribute;
import br.ufsc.tcc.gamifyEngine.service.AttributeService;
import br.ufsc.tcc.gamifyEngine.service.BadgeService;
import br.ufsc.tcc.gamifyEngine.service.LogService;
import br.ufsc.tcc.gamifyEngine.service.RuleService;
import br.ufsc.tcc.gamifyEngine.service.UserService;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
@RequestMapping("/api")
public class RestApiController {

	@Autowired
	UserService userService;
```

```
	@Autowired
	RuleService ruleService;

	@Autowired
	AttributeService attributeService;

	@Autowired
	BadgeService badgeService;

	@Autowired
	LogService logService;

public static final Logger logger =
LoggerFactory.getLogger(RestApiController.class);

	/**
	 *
	 *  ------------------------------------ USER
----------------------------------
	 *
	 * **/

	@RequestMapping(value = "/users/", method =
RequestMethod.GET)
	public ResponseEntity<?> listAllUsers() {
		List<User> usersList = new ArrayList<User>();
		Iterable<User> users = userService.findAllUsers();

		for (User user : users) {
			usersList.add(user);
		}

		if (usersList.size() == 0) {
			return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<>(usersList, HttpStatus.OK);
	}

	@RequestMapping(value = "/users/{userId}", method =
RequestMethod.GET)
```

```
	public ResponseEntity<?> getUser(@PathVariable int userId) {
		User user = userService.getUser(userId);

		if (user == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}

		return new ResponseEntity<>(user, HttpStatus.OK);
	}

	@RequestMapping(value = "/users/", method =
RequestMethod.POST)
	public ResponseEntity<?> postUser(@RequestBody User user) {

		user.setAttributes(new ArrayList<>());
		user.setBadges(new ArrayList<>());

		User newUser = this.userService.saveUser(user);
		return new ResponseEntity<>(newUser, HttpStatus.OK);
	}

	@RequestMapping(value = "/users/{userId}", method =
RequestMethod.PUT)
public ResponseEntity<?> putUser(@PathVariable int userId,
@RequestBody User user) {

		logger.info("Updating User with id {}", userId);

		User currentUser = userService.getUser(userId);

		if (currentUser == null) {
		logger.error("Unable to update. User with id {} not
found.", userId);
	return new ResponseEntity<>("Unable to update. User with id "
+ userId + " not found.", HttpStatus.NOT_FOUND);
		}

		currentUser.setActive(user.isActive());
		currentUser.setXp(user.getXp());
		currentUser.setLevel(user.getLevel());
		currentUser.setCurrentXp(user.getCurrentXp());
```

```
//tratamento necessário para que o Hibernate não perca referência
do attribute

          currentUser.getAttributes().clear();

currentUser.getAttributes().addAll(user.getAttributes());
          currentUser.setAttributes(currentUser.getAttributes());

          currentUser.setBadges(user.getBadges());

          currentUser.getAttributes().stream().forEach(attUser ->
{
attUser.setId(new RuleAttributeKey(userId,
attUser.getAttribute().getId()));
          });

        this.userService.saveUser(currentUser);
        return new ResponseEntity<User>(currentUser,
HttpStatus.OK);
     }

@RequestMapping(value = "/users/{userId}", method =
RequestMethod.DELETE)
 public ResponseEntity<?> deleteUser(@PathVariable int userId) {
          try {
               userService.deleteUser(userId);
          } catch (Exception e) {
               // TODO: handle exception
          }

          return new ResponseEntity<>(HttpStatus.OK);
     }

     /**
      *
      *  ------------------------------------ ATTRIBUTES
----------------------------------
      *
      * **/
```

```
	@RequestMapping(value = "/attributes/", method =
RequestMethod.GET)
	public ResponseEntity<?> getAttributes() {

		try {
			Iterable<Attribute> atts =
attributeService.findAllAttributes();

			List<Attribute> attrList = new
ArrayList<Attribute>();

			for (Attribute attribute : atts) {
				attrList.add(attribute);
			}


			return new ResponseEntity<>(attrList,
HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@RequestMapping(value = "/attributes/{attributeId}", method =
RequestMethod.GET)
	public ResponseEntity<?> getAttribute(@PathVariable int
attributeId) {
		Attribute att =
attributeService.getAttribute(attributeId);
		if (att == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(att, HttpStatus.OK);
	}

	@RequestMapping(value = "/attributes/", method =
RequestMethod.POST)
	public ResponseEntity<?> postAttribute(@RequestBody Attribute
attribute) {
```

```
            Attribute newAttribute =
this.attributeService.saveAttribute(attribute);
            return new ResponseEntity<>(newAttribute,
HttpStatus.OK);
      }

      @RequestMapping(value = "/attributes/{attributeId}", method =
RequestMethod.PUT)
      public ResponseEntity<?> putAttribute(@PathVariable int
attributeId, @RequestBody Attribute attribute) {
            Attribute currentAttribute =
this.attributeService.getAttribute(attributeId);

            if (currentAttribute == null) {
                  logger.error("Unable to update. Attribute with id
{} not found.", attributeId);
                return new ResponseEntity<>("Unable to update.
Attribute with id " + attributeId + " not found.",
HttpStatus.NOT_FOUND);
            }

currentAttribute.setDescription(attribute.getDescription());
            currentAttribute.setName(attribute.getName());

        this.attributeService.saveAttribute(currentAttribute);
        return new ResponseEntity<Attribute>(currentAttribute,
HttpStatus.OK);
      }

      @RequestMapping(value = "/attributes/{attributeId}", method =
RequestMethod.DELETE)
      public ResponseEntity<?> deleteAttribute(@PathVariable int
attributeId) {
            try {
                  attributeService.deleteAttribute(attributeId);
            } catch (Exception e) {
                  return new ResponseEntity<>(e,
HttpStatus.BAD_GATEWAY);
            }

            return new ResponseEntity<>(HttpStatus.OK);
```

```
	}

	/**
	 *
	 *  ------------------------------------ BADGES
----------------------------------
	 *
	 **/

	@RequestMapping(value = "/badges/", method =
RequestMethod.GET)
	public ResponseEntity<?> getbadges() {

		try {
			Iterable<Badge> badgesList =
badgeService.findAllbadges();

			List<Badge> attrList = new ArrayList<Badge>();

			for (Badge badge : badgesList) {
				attrList.add(badge);
			}


			return new ResponseEntity<>(attrList,
HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@RequestMapping(value = "/badges/{badgeId}", method =
RequestMethod.GET)
	public ResponseEntity<?> getBadge(@PathVariable int badgeId)
{
		Badge att = badgeService.getBadge(badgeId);
		if (att == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(att, HttpStatus.OK);
```

```
	}

	@RequestMapping(value = "/badges/", method =
RequestMethod.POST)
	public ResponseEntity<?> postBadge(@RequestBody Badge badge)
{

		Badge newBadge = this.badgeService.saveBadge(badge);
		return new ResponseEntity<>(newBadge, HttpStatus.OK);
	}

	@RequestMapping(value = "/badges/{badgeId}", method =
RequestMethod.PUT)
	public ResponseEntity<?> putBadge(@PathVariable int badgeId,
@RequestBody Badge badge) {
		Badge currentBadge =
this.badgeService.getBadge(badgeId);

		if (currentBadge == null) {
			logger.error("Unable to update. Badge with id {}
not found.", badgeId);
		    return new ResponseEntity<>("Unable to update.
Badge with id " + badgeId + " not found.", HttpStatus.NOT_FOUND);
		}
		currentBadge.setDescription(badge.getDescription());
		currentBadge.setName(badge.getName());

	    this.badgeService.saveBadge(currentBadge);
	    return new ResponseEntity<Badge>(currentBadge,
HttpStatus.OK);
	}

	@RequestMapping(value = "/badges/{badgeId}", method =
RequestMethod.DELETE)
	public ResponseEntity<?> deleteBadge(@PathVariable int
badgeId) {
		try {
			badgeService.deleteBadge(badgeId);
		} catch (Exception e) {
			return new ResponseEntity<>(e,
HttpStatus.BAD_GATEWAY);
```

```
            }

            return new ResponseEntity<>(HttpStatus.OK);
      }


      /**
       *
       *  ------------------------------------ RULES
----------------------------------
       *
       **/

      @RequestMapping(value = "/rules/", method =
RequestMethod.GET)
      public ResponseEntity<?> getRules() {
            List<Rule> rulesList = new ArrayList<Rule>();
            Iterable<Rule> rules = ruleService.findAllRules();

            for (Rule rule : rules) {
                  rulesList.add(rule);
            }

            System.out.println(rulesList);

            if (rulesList.size() == 0) {
                  return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(rulesList, HttpStatus.OK);
      }

      @RequestMapping(value = "/rules/{ruleId}", method =
RequestMethod.GET)
      public ResponseEntity<?> getRule(@PathVariable int ruleId) {
            Rule rule = ruleService.getRule(ruleId);

            if (rule == null) {
                  return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
            return new ResponseEntity<>(rule, HttpStatus.OK);
```

```
	}


	@RequestMapping(value = "/rules/type/{type}", method =
RequestMethod.GET)
	public ResponseEntity<?> getRuleByType(@PathVariable String
type) {
		List<Rule> rule = ruleService.getRuleByType(type);

		if (rule == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(rule, HttpStatus.OK);
	}

	@RequestMapping(value = "/rules/", method =
RequestMethod.POST)
	public ResponseEntity<?> postRule(@RequestBody Rule rule) {

		Rule newRule = this.ruleService.saveRule(rule);
		return new ResponseEntity<>(newRule, HttpStatus.OK);
	}

	@RequestMapping(value = "/rules/{ruleId}", method =
RequestMethod.PUT)
	public ResponseEntity<?> putRule(@PathVariable int ruleId,
@RequestBody Rule rule) {
		Rule currentRule = this.ruleService.getRule(ruleId);

		if (currentRule == null) {
			logger.error("Unable to update. Rule with id {}
not found.", ruleId);
			return new ResponseEntity<>("Unable to update. Rule
with id " + ruleId + " not found.", HttpStatus.NOT_FOUND);
		}

		currentRule.setName(rule.getName());
		currentRule.setActive(rule.isActive());
		currentRule.setFinished(rule.isFinished());

currentRule.setTimesToComplete(rule.getTimesToComplete());
```

```
            currentRule.setType(rule.getType());
            currentRule.setXp(rule.getXp());

          this.ruleService.saveRule(currentRule);
          return new ResponseEntity<Rule>(currentRule,
HttpStatus.OK);
      }


      @RequestMapping(value = "/rules/{ruleId}", method =
RequestMethod.DELETE)
      public ResponseEntity<?> deleteRule(@PathVariable int ruleId)
{
            try {
                  ruleService.deleteRule(ruleId);
            } catch (Exception e) {
                  // TODO: handle exception
            }
            return new ResponseEntity<>(HttpStatus.OK);
      }


      /**
       *
       *  ------------------------------------ USER BADGES
----------------------------------
       *
       **/
      @RequestMapping(value = "/badges/{badgeId}/users/{userId}",
method = RequestMethod.DELETE)
      public ResponseEntity<?> deleteUserBadge(@PathVariable int
badgeId, @PathVariable int userId) {
            try {
                  this.userService.deleteUserBadge(userId, badgeId);
            } catch (Exception e) {
                  // TODO: handle exception
            }
            return new ResponseEntity<>(HttpStatus.OK);
      }

      @RequestMapping(value = "/badges/{badgeId}/users/{userId}",
```

```
method = RequestMethod.POST)
	public ResponseEntity<?> putUserBadge(@PathVariable int
badgeId, @PathVariable int userId) {
		try {
			User user = this.userService.getUser(userId);


if(!user.getBadges().contains(this.badgeService.getBadge(badgeId))
){

user.getBadges().add(this.badgeService.getBadge(badgeId));
			}

			this.userService.saveUser(user);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}


	/**
	 *
	 *  ------------------------------------ LOGS
----------------------------------
	 *
	 **/

	@RequestMapping(value = "/logs/{logId}", method =
RequestMethod.GET)
	public ResponseEntity<?> getLog(@PathVariable int logId) {


		LogEvent log = this.logService.getLog(logId);

		if (log == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(log, HttpStatus.OK);
	}
```

```
	@RequestMapping(value = "/logs/users/{userId}", method =
RequestMethod.GET)
	public ResponseEntity<?> getLogs(@PathVariable int userId) {
		try {
			List<LogEvent> logs =
this.logService.getUserLogs(userId);
			if( logs != null)
				return new ResponseEntity<>(logs,
HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@RequestMapping(value = "/logs/", method =
RequestMethod.POST)
	public ResponseEntity<?> createLog(@RequestBody LogEvent
logEvent) {
		try {
			LogEvent newLog =
this.logService.saveLog(logEvent);
			if(newLog != null)
				return new ResponseEntity<>(newLog,
HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@RequestMapping(value = "/logs/{logId}", method =
RequestMethod.PUT)
	public ResponseEntity<?> updateLog(@RequestBody LogEvent log,
@PathVariable int logId) {
		try {
			LogEvent currentLog =
this.logService.getLog(logId);
			currentLog.setDateEvent(log.getDateEvent());
			currentLog.setRule(log.getRule());
			currentLog.setUser(log.getUser());
```

```
                LogEvent logUpdated =
this.logService.saveLog(currentLog);

                if(logUpdated != null)
                      return new ResponseEntity<>(logUpdated,
HttpStatus.OK);

           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     @RequestMapping(value =
"/logs/rules/{ruleId}/users/{userId}", method =
RequestMethod.DELETE)
     public ResponseEntity<?> deleteUserLog(@PathVariable int
ruleId, @PathVariable int userId) {
           try {
                this.logService.deleteRuleLogsFromUser(ruleId,
userId);
           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     @RequestMapping(value = "/logs/{logId}", method =
RequestMethod.DELETE)
     public ResponseEntity<?> deleteLog(@PathVariable int logId) {

           try {
                logService.deleteLog(logId);
           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }


     /**
```

```
       *
       *  ------------------------------------ RULE ATTRIBUTES
-----------------------------------
       *
       **/

      @RequestMapping(value = "/attributes/rules", method =
RequestMethod.GET)
      public ResponseEntity<?> getAllAttributeRules() {
            List<RuleAttribute> ruleAttributeList = new
ArrayList<RuleAttribute>();
            Iterable<RuleAttribute> rules =
ruleService.findAllAttributeRules();

            for (RuleAttribute rule : rules) {
                  ruleAttributeList.add(rule);
            }

            if (ruleAttributeList.size() == 0) {
                  return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(ruleAttributeList,
HttpStatus.OK);
      }

      @RequestMapping(value =
"/attributes/rules/{ruleAttributeId}", method = RequestMethod.GET)
      public ResponseEntity<?> getRuleAttribute(@PathVariable int
ruleAttributeId) {
            RuleAttribute ruleAtt =
ruleService.getRuleAttribute(ruleAttributeId);

            if (ruleAtt == null) {
                  return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
            return new ResponseEntity<>(ruleAtt, HttpStatus.OK);
      }

      @RequestMapping(value = "/attributes/rules", method =
RequestMethod.POST)
```

```
	public ResponseEntity<?> createRuleAttribute(@RequestBody
RuleAttribute ruleAttribute) {
		try {
			RuleAttribute newRuleAttribute =
this.ruleService.saveRuleAttribute(ruleAttribute);
			if(newRuleAttribute != null)
				return new
ResponseEntity<>(newRuleAttribute, HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}


	@RequestMapping(value =
"/attributes/rules/{ruleAttributeId}", method =
RequestMethod.DELETE)
	public ResponseEntity<?> deleteRuleAttribute(@PathVariable
int ruleAttributeId) {
		try {
			ruleService.deleteRuleAttribute(ruleAttributeId);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@RequestMapping(value = "/attributes/rules/byRule/{ruleId}",
method = RequestMethod.GET)
	public ResponseEntity<?>
getRuleAttributesByRule(@PathVariable int ruleId) {

		List<RuleAttribute> ruleAttributeList = new
ArrayList<RuleAttribute>();
		Iterable<RuleAttribute> rules =
ruleService.getRuleAttributeByRule(ruleId);

//		Iterable<RuleAttribute> rules =
ruleService.findAllAttributeRules();
```

```
            for (RuleAttribute rule : rules) {
                  ruleAttributeList.add(rule);
            }

            if (ruleAttributeList.size() == 0) {
                  return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(ruleAttributeList,
HttpStatus.OK);
      }


      /**
       *
       *  ------------------------------------ RULE BADGES
----------------------------------
       *
       **/


      @RequestMapping(value = "/badges/rules", method =
RequestMethod.GET)
      public ResponseEntity<?> getAllBadgeRules() {
            List<RuleBadge> ruleBadgeList = new
ArrayList<RuleBadge>();
            Iterable<RuleBadge> rules =
ruleService.findAllBadgeRules();

            for (RuleBadge rule : rules) {
                  ruleBadgeList.add(rule);
            }

            if (ruleBadgeList.size() == 0) {
                  return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(ruleBadgeList,
HttpStatus.OK);
      }
```

```
	@RequestMapping(value = "/badges/rules/{ruleBadgeId}", method
= RequestMethod.GET)
	public ResponseEntity<?> getRuleBadge(@PathVariable int
ruleBadgeId) {
		RuleBadge ruleBadge =
ruleService.getRuleBadge(ruleBadgeId);

		if (ruleBadge == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(ruleBadge, HttpStatus.OK);
	}

	@RequestMapping(value = "/badges/rules", method =
RequestMethod.POST)
	public ResponseEntity<?> createRuleBadge(@RequestBody
RuleBadge ruleBadge) {
		try {
			RuleBadge newRuleBadge =
this.ruleService.saveRuleBadge(ruleBadge);
			if(newRuleBadge != null)
				return new ResponseEntity<>(newRuleBadge,
HttpStatus.OK);
		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
	}


	@RequestMapping(value = "/badges/rules/{ruleBadgeId}", method
= RequestMethod.PUT)
	public ResponseEntity<?> updateLog(@RequestBody RuleBadge
ruleBadge, @PathVariable int ruleBadgeId) {
		try {
			RuleBadge currentRuleBadge =
this.ruleService.getRuleBadge(ruleBadgeId);
			currentRuleBadge.setBadge(ruleBadge.getBadge());
			currentRuleBadge.setRule(ruleBadge.getRule());
			RuleBadge ruleBadgeUpdated =
this.ruleService.saveRuleBadge(currentRuleBadge);
```

```
                if(ruleBadgeUpdated != null)
                     return new
ResponseEntity<>(ruleBadgeUpdated, HttpStatus.OK);

           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     @RequestMapping(value = "/badges/rules/{ruleBadgeId}", method
= RequestMethod.DELETE)
     public ResponseEntity<?> deleteRuleBadge(@PathVariable int
ruleBadgeId) {
           try {
                ruleService.deleteRuleBadge(ruleBadgeId);
           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     /**
      *
      *  ------------------------------------ RULE LEVEL
----------------------------------
      *
      **/

     @RequestMapping(value = "/level/rules/", method =
RequestMethod.GET)
     public ResponseEntity<?> getAllRuleLevel() {
           List<RuleLevel> ruleLevelList = new
ArrayList<RuleLevel>();
           Iterable<RuleLevel> rules =
ruleService.findAllLevelRules();

           for (RuleLevel rule : rules) {
                ruleLevelList.add(rule);
           }
```

```
            if (ruleLevelList.size() == 0) {
                  return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(ruleLevelList,
HttpStatus.OK);
      }


      @RequestMapping(value = "/level/rules/{ruleLevelId}", method
= RequestMethod.GET)
      public ResponseEntity<?> getRuleLevel(@PathVariable int
ruleLevelId) {
            RuleLevel ruleLevel =
ruleService.getRuleLevel(ruleLevelId);

            if (ruleLevel == null) {
                  return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
            return new ResponseEntity<>(ruleLevel, HttpStatus.OK);
      }

      @RequestMapping(value = "/level/rules/{ruleLevelId}", method
= RequestMethod.DELETE)
      public ResponseEntity<?> deleteRuleLevel(@PathVariable int
ruleLevelId) {
            try {
                  ruleService.deleteRuleLevel(ruleLevelId);
            } catch (Exception e) {
                  // TODO: handle exception
            }
            return new ResponseEntity<>(HttpStatus.OK);
      }

      @RequestMapping(value = "/level/rules/", method =
RequestMethod.POST)
      public ResponseEntity<?> insertRuleLevel(@RequestBody
RuleLevel ruleLevel) {
            try {
```

```
                RuleLevel newRuleLevel =
ruleService.saveRuleLevel(ruleLevel);
                if(newRuleLevel != null)
                      return new ResponseEntity<>(newRuleLevel,
HttpStatus.OK);

           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     @RequestMapping(value = "/level/rules/{ruleLevelId}", method
= RequestMethod.PUT)
     public ResponseEntity<?> updateLog(@RequestBody RuleLevel
ruleLevel, @PathVariable int ruleLevelId) {
           try {
                RuleLevel currentRuleLevel =
this.ruleService.getRuleLevel(ruleLevelId);

currentRuleLevel.setEndLevel(ruleLevel.getEndLevel());
                currentRuleLevel.setRule(ruleLevel.getRule());

currentRuleLevel.setStartLevel(ruleLevel.getStartLevel());

currentRuleLevel.setXpToLevelUp(ruleLevel.getXpToLevelUp());

                RuleLevel ruleLevelUpdated =
this.ruleService.saveRuleLevel(currentRuleLevel);

                if(ruleLevelUpdated != null)
                      return new
ResponseEntity<>(ruleLevelUpdated, HttpStatus.OK);

           } catch (Exception e) {
                // TODO: handle exception
           }
           return new ResponseEntity<>(HttpStatus.OK);
     }

     /**
```

```
      *
      *  ------------------------------------ LEVEL REWARD
----------------------------------
      *
      **/

//    @RequestMapping(value = "/level/rewards/{levelRewardId}",
method = RequestMethod.GET)
//    public ResponseEntity<?> getLevelReward(@PathVariable int
levelRewardId) {
//         LevelReward levelReward =
ruleService.getLevelReward(levelRewardId);
//
//         if (levelReward == null) {
//              return new ResponseEntity<>(HttpStatus.NOT_FOUND);
//         }
//         return new ResponseEntity<>(levelReward,
HttpStatus.OK);
//    }

     @RequestMapping(value = "/level/rewards/", method =
RequestMethod.POST)
     public ResponseEntity<?> insertRuleLevel(@RequestBody
LevelReward levelReward) {
          try {
//              LevelRewardKey lrk = new
LevelRewardKey(levelReward.getAttribute().getId(),
levelReward.getRuleLevel().getId());
//              levelReward.setId(lrk);

                LevelReward newLevelReward =
ruleService.saveLevelReward(levelReward);
                if(newLevelReward != null)
                     return new ResponseEntity<>(newLevelReward,
HttpStatus.OK);

          } catch (Exception e) {
                // TODO: handle exception
          }
          return new ResponseEntity<>(HttpStatus.OK);
     }
```

```
	@RequestMapping(value = "/level/rewards/rules/{idRuleLevel}",
method = RequestMethod.GET)
	public ResponseEntity<?>
getLevelRewardsByRuleLevel(@PathVariable int idRuleLevel) {

		List<LevelReward> levelRewardList = new
ArrayList<LevelReward>();
		Iterable<LevelReward> rulesLoaded =
ruleService.getLevelRewardsByRuleLevel(idRuleLevel);

//		Iterable<RuleAttribute> rules =
ruleService.findAllAttributeRules();

		for (LevelReward levelReward : rulesLoaded) {
			levelRewardList.add(levelReward);
		}

		if (levelRewardList.size() == 0) {
			return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<>(levelRewardList,
HttpStatus.OK);
	}

@RequestMapping(value = "/badges/attributes/rewards/", method =
RequestMethod.POST)
public ResponseEntity<?> insertRuleBadgeAttribute(@RequestBody
RuleBadgeAttribute ruleBadgeAttribute) {
		try {
			RuleBadgeAttribute newuleBadgeAttribute =
ruleService.saveRuleBadgeAttribute(ruleBadgeAttribute);
			if(newuleBadgeAttribute != null)
				return new
ResponseEntity<>(newuleBadgeAttribute, HttpStatus.OK);

		} catch (Exception e) {
			// TODO: handle exception
		}
		return new ResponseEntity<>(HttpStatus.OK);
```

```
	}

@RequestMapping(value = "/event-rule/{ruleId}/{userId}", method =
RequestMethod.GET)
	public ResponseEntity<?> processEvent(@PathVariable int
ruleId, @PathVariable int userId) {

		Map< String, Object > response =
this.ruleService.processRule(userId, ruleId);

		return new ResponseEntity<>(response, HttpStatus.OK);
	}
}
```

AttributeDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import org.springframework.data.repository.CrudRepository;

import br.ufsc.tcc.gamifyEngine.model.Attribute;

public interface AttributeDao  extends CrudRepository<Attribute,
Integer> {
	public Attribute findById(int attributeId);
}
```

BadgeDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import org.springframework.data.repository.CrudRepository;

import br.ufsc.tcc.gamifyEngine.model.Badge;

public interface BadgeDao  extends CrudRepository<Badge, Integer>
```

```
{
	public Badge findById(int badgeId);
}
```

LevelRewardDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import br.ufsc.tcc.gamifyEngine.model.LevelReward;

public interface LevelRewardDao  extends
CrudRepository<LevelReward, Integer> {

	@Query("select r from LevelReward r where r.ruleLevel.id =
?1")
	public List<LevelReward> findCurrentLevelReward(int
levelRewardId);


	@Query("select r from LevelReward r where r.ruleLevel.id =
?1")
	public Iterable<LevelReward> getLevelRewardsByRuleLevel(int
idRuleLevel);
}
```

LogEventDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.transaction.annotation.Transactional;

import br.ufsc.tcc.gamifyEngine.model.LogEvent;

public interface LogEventDao extends CrudRepository<LogEvent,
Integer> {
//    public LogEvent fingLogByUserAndRule (int userId, int
ruleId);

      @Query(value= "select * from Log_event log where log.rule_id
= ?2 AND log.user_id = ?1", nativeQuery=true)
      public List<LogEvent> findByUserAndRule(int userId, int
ruleId);

      @Query("select log from LogEvent log where log.id = ?1")
      public LogEvent findById(int logId);

      @Query("select log from LogEvent log where log.user.id = ?1")
      public List<LogEvent> getUserLogs(int userId);

      @Transactional
      @Modifying
      @Query(value= "DELETE FROM log_event where rule_id = ?1 AND
user_id = ?2", nativeQuery=true)
      public void deleteRuleLogsFromUser(int ruleId, int userId);

}
```

RuleAttributeDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;

import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
```

```
@Transactional
public interface RuleAttributeDao extends
CrudRepository<RuleAttribute, Integer> {

  /**
   * This method will find an User instance in the database by its
email.
   * Note that this method is not implemented and its working code
will be
   * automagically generated from its signature by Spring Data
JPA.
   */

      public RuleAttribute findById(int ruleAttributeId);

      @Query("select r from RuleAttribute r where r.rule.id = ?1")
      public List<RuleAttribute> findRuleAttributeByRuleId(int
ruleId);
}
```

RuleBadgeAttributeDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

import br.ufsc.tcc.gamifyEngine.model.RuleBadgeAttribute;

public interface RuleBadgeAttributeDao  extends
CrudRepository<RuleBadgeAttribute, Integer> {

      public List<RuleBadgeAttribute> findByAttributeId(int
attributeId);
}
```

RuleBadgeDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;

import br.ufsc.tcc.gamifyEngine.model.RuleBadge;
import br.ufsc.tcc.gamifyEngine.model.RuleBadgeAttribute;

@Transactional
public interface RuleBadgeDao extends CrudRepository<RuleBadge,
Integer> {

  /**
   * This method will find an User instance in the database by its
email.
   * Note that this method is not implemented and its working code
will be
   * automagically generated from its signature by Spring Data
JPA.
   */

      public RuleBadge findById(int ruleAttributeId);

      @Query("select r from RuleBadge r where r.rule.id = ?1")
      public RuleBadge findRuleBadgeByRuleId(int ruleId);

      public List<RuleBadge> findByAttributeId(int attributeId);
}
```

RuleDao.java

---

package br.ufsc.tcc.gamifyEngine.dao;

```
import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;
```

```
import br.ufsc.tcc.gamifyEngine.model.Rule;

@Transactional
public interface RuleDao extends CrudRepository<Rule, Integer> {

  /**
   * This method will find an User instance in the database by its
email.
   * Note that this method is not implemented and its working code
will be
   * automagically generated from its signature by Spring Data
JPA.
   */
  public Rule findById(int ruleId);

  @Query(value="select * from rule r where r.type = ?1",
nativeQuery=true)
  public List<Rule> findRuleByType(String type);

}
```

RuleLevelDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;

import br.ufsc.tcc.gamifyEngine.model.Rule;
import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleLevel;

@Transactional
public interface RuleLevelDao extends CrudRepository<RuleLevel,
Integer> {

  /**
   * This method will find an User instance in the database by its
email.
```

```
   * Note that this method is not implemented and its working code
will be
   * automagically generated from its signature by Spring Data
JPA.
   */

      public RuleLevel findById(int ruleLevelId);

      @Query("select r from RuleLevel r where r.startLevel <= ?1
and r.endLevel >= ?1")
      public RuleLevel findByLevelRange(int currentUserLevel);

      @Query(value="select coalesce(max(r.end_level),0) as maxLevel
from rule_level r", nativeQuery=true)
      public int findHighestLevelRange();
}
```

UserDao.java

```
package br.ufsc.tcc.gamifyEngine.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.transaction.annotation.Transactional;

import br.ufsc.tcc.gamifyEngine.model.LevelReward;
import br.ufsc.tcc.gamifyEngine.model.User;

@Transactional
public interface UserDao extends CrudRepository<User, Integer> {

  /**
   * This method will find an User instance in the database by its
email.
   * Note that this method is not implemented and its working code
will be
   * automagically generated from its signature by Spring Data
```

```
JPA.
   */
  public User findById(int userId);

  @Modifying
  @Query(value = "DELETE FROM user_badges where user_id = ?1 and
badge_id = ?2", nativeQuery = true)
  public void deleteUserBadge(int userId, int badgeId);

}
```

Attribute.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;

@Entity
public class Attribute {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	@NotNull
	private String name;

	private String description;

	public Attribute() {
		// TODO Auto-generated constructor stub
	}

	public int getId() {
		return id;
	}
```

```
	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}
}
```

Badge.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;

@Entity
public class Badge {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	@NotNull
	private String name;
```

```
    private String description;

    public Badge() {

    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
public boolean equals(Object object)
{
    boolean sameSame = false;

    if (object != null && object instanceof Badge)
    {
        sameSame = this.id == ((Badge) object).id;
    }

    return sameSame;
```

```
    }
}
```

Engine.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import java.sql.Timestamp;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;

import br.ufsc.tcc.gamifyEngine.dao.LevelRewardDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleBadgeDao;
import br.ufsc.tcc.gamifyEngine.service.AttributeService;
import br.ufsc.tcc.gamifyEngine.service.BadgeService;
import br.ufsc.tcc.gamifyEngine.service.LogService;
import br.ufsc.tcc.gamifyEngine.service.RuleService;
import br.ufsc.tcc.gamifyEngine.service.UserService;


public class Engine {

	@Autowired
	UserService userService;

	@Autowired
	RuleService ruleService;

	@Autowired
	AttributeService attributeService;

	@Autowired
	BadgeService badgeService;

	@Autowired
	LogService logService;
```

```
	LevelRewardDao levelRewardDao;

	RuleBadgeDao ruleBadgeDao;

	public Engine() {

	}

	public Map<String, Object> processEvent(int ruleId, int
userId) {
		Rule rule = this.ruleService.getRule(ruleId);
		User user = userService.getUser(userId);
		String ruleType = rule.getType();

		Map< String, Map<Attribute, Integer> > diff = new
HashMap< String, Map<Attribute, Integer> >();

		boolean completed = false;

		List<LogEvent> logs =
this.logService.getLogByUserAndRule(userId, ruleId);
		int timesCompleted = logs.size();

		if(rule.getTimesToComplete() == timesCompleted) {
			completed = true;
		}

		User userUpdated = null;

		if(!completed) {

			timesCompleted++;

			//TODO ruleType troca pra enum depois
			switch (ruleType) {
			case "attribute":
List<RuleAttribute> ruleAttributes = null;
			ruleAttributes =
ruleService.getRuleAttributeByRule(ruleId);
```

```
Map<Attribute, Integer> attributesMap = new HashMap<Attribute,
Integer>();

                    //atualiza os atributos do usuário
                    for (RuleAttribute ruleAttribute :
ruleAttributes) {
                         int amount = ruleAttribute.getAmount();
                         int attId =
ruleAttribute.getAttribute().getId();

                         user.getAttributes().stream()
                                   .filter(attribute ->
attribute.getAttribute().getId() == attId)
                                   .forEach( attribute -> {

attribute.setValue(attribute.getValue() + amount);

attributesMap.put(attribute.getAttribute(), amount);
                                   });

                    }

                    diff.put("attributes", attributesMap);

                    break;
               case "badge":
                    RuleBadge ruleBadge = null;
                    ruleBadge =
ruleService.getRuleBadgesByRule(ruleId);
                    if(timesCompleted >=
rule.getTimesToComplete()) {

user.getBadges().add(ruleBadge.getBadge());
                         this.ruleService.evaluate("badge",
user, null);
                    }
                    break;
               default:
                    break;
               }
```

```
                //sempre vai ganhar XP, mesmo que a rule não for
totalmente completa
                user.setXp(user.getXp() + rule.getXp());
                user.setCurrentXp(user.getCurrentXp() +
rule.getXp());

                this.ruleService.evaluate("xp", user, null);

                userUpdated = userService.saveUser(user);

                //faz log do evento
                LogEvent logEvent = new LogEvent();
                logEvent.setRule(rule.getId());
                logEvent.setUser(user.getId());
                Timestamp timestamp = new
Timestamp(System.currentTimeMillis());
                logEvent.setDateEvent(timestamp);
                this.logService.insertLog(logEvent);
           }



//         example.put("newLevel", "2");

           Map<String, Object> response = new HashMap<String,
Object>();

           response.put("user", userUpdated);
           response.put("diff", diff);

           return response;
     }

     public void evaluate(String type, User user, Attribute att) {
//TODO mudar pra enum

           RuleLevel ruleLevel = null;

           switch (type) {
                case "xp":
```

```
                        ruleLevel =
this.ruleService.findAdequatedRuleLevel(user);

                        int currentUserXp = user.getCurrentXp();

                        if(ruleLevel != null) {
                              while(currentUserXp >=
ruleLevel.getXpToLevelUp()) {
                                    user.setLevel(user.getLevel() +
1);

user.setCurrentXp(user.getCurrentXp() -
ruleLevel.getXpToLevelUp());
                                    currentUserXp =
user.getCurrentXp();
                                    this.evaluate("level", user,
null);
                                    ruleLevel =
this.ruleService.findAdequatedRuleLevel(user);
                              }
                        }
                        break;
                  case "badge":
                        //TODO asdasd
                        break;
                  case "attribute":
                        //List<RuleBadgeAttribute> ruleBadgesAtt =
this.ruleBadgeAttributeDao.findByAttributeId(att.getId());
                        List<RuleBadge> ruleBadgesAtt =
this.ruleBadgeDao.findByAttributeId(att.getId());

                        for(RuleBadge ruleBadgeAtt: ruleBadgesAtt){

                              for(UserAttribute userAttribute:
user.getAttributes()) {

if(userAttribute.getAttribute().getId() ==
ruleBadgeAtt.getAttribute().getId() && userAttribute.getValue() >=
ruleBadgeAtt.getGoalValue())
                                    {
                                          //TODO verificar se está no
```

```
LOG

user.getBadges().add(ruleBadgeAtt.getBadge());
//
ruleBadgeAtt.getRule().setFinished(true);

this.ruleBadgeDao.save(ruleBadgeAtt);
                                    }
                               }
                          }
                          break;
                     case "level":
                          ruleLevel =
this.ruleService.findAdequatedRuleLevel(user);
                          List<LevelReward> levelRewards =
this.levelRewardDao.findCurrentLevelReward(ruleLevel.getId());

                          //atualiza os atributos do usuário
                          for (LevelReward levelReward : levelRewards)
{
                               int amount = levelReward.getAmount();
                               int attId =
levelReward.getAttribute().getId();

                               user.getAttributes().stream()
                                         .filter(attribute ->
attribute.getAttribute().getId() == attId)
                                         .forEach(attribute ->
attribute.setValue(attribute.getValue() + amount));


                               this.evaluate("attribute", user,
levelReward.getAttribute());
                          }
                          break;
                     default:
                          break;
               }
          }
}
```

FeedbackAttributes.Java

```
package br.ufsc.tcc.gamifyEngine.model;

public class FeedbackAttributes {

	private Attribute attribute;
	private int amountChanged;

	public FeedbackAttributes() {
	}

	public FeedbackAttributes(Attribute att, int amountChanged) {
		this.attribute = att;
		this.amountChanged = amountChanged;
	}

	public Attribute getAttribute() {
		return attribute;
	}

	public void setAttribute(Attribute attribute) {
		this.attribute = attribute;
	}

	public int getAmountChanged() {
		return amountChanged;
	}

	public void setAmountChanged(int amountChanged) {
		this.amountChanged = amountChanged;
	}
}
```

FeedbackLevel.Java

```
package br.ufsc.tcc.gamifyEngine.model;
```

```java
public class FeedbackLevel {
	private boolean levelIncreased;
	private int newLevel;

	public FeedbackLevel(boolean levelIncreased, int newLevel) {
		this.levelIncreased = levelIncreased;
		this.newLevel = newLevel;
	}

	public FeedbackLevel() {
		// TODO Auto-generated constructor stub
	}

	public boolean isLevelIncreased() {
		return levelIncreased;
	}

	public void setLevelIncreased(boolean levelIncreased) {
		this.levelIncreased = levelIncreased;
	}

	public int getNewLevel() {
		return newLevel;
	}

	public void setNewLevel(int newLevel) {
		this.newLevel = newLevel;
	}

}
```

FeedbackXp.Java

```java
package br.ufsc.tcc.gamifyEngine.model;

public class FeedbackXp {
	private boolean xpChanged;
	private int amountChanged;
	private int userTotalXp;
```

```
	private int userNewCurrentXp;

	public FeedbackXp(boolean xpChanged, int amountChanged, int
newXp, int newCurrentXp) {
		this.xpChanged = xpChanged;
		this.amountChanged = amountChanged;
		this.userTotalXp = newXp;
		this.userNewCurrentXp = newCurrentXp;
	}

	public FeedbackXp() {
		// TODO Auto-generated constructor stub
	}

	public boolean isXpChanged() {
		return xpChanged;
	}

	public void setXpChanged(boolean xpChanged) {
		this.xpChanged = xpChanged;
	}

	public int getAmountChanged() {
		return amountChanged;
	}

	public void setAmountChanged(int amountChanged) {
		this.amountChanged = amountChanged;
	}

	public int getUserTotalXp() {
		return userTotalXp;
	}

	public void setUserTotalXp(int userTotalXp) {
		this.userTotalXp = userTotalXp;
	}

	public int getUserNewCurrentXp() {
		return userNewCurrentXp;
	}
```

```
	public void setUserNewCurrentXp(int userNewCurrentXp) {
		this.userNewCurrentXp = userNewCurrentXp;
	}
}
```

LevelReward.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.MapsId;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonBackReference;

import br.ufsc.tcc.gamifyEngine.compositeKeys.LevelRewardKey;

@Entity
@Table(name="level_reward")
public class LevelReward {


	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

//	@MapsId("attribute")
	@ManyToOne
	private Attribute attribute;

//	@MapsId("ruleLevel")
	@ManyToOne
```

```
	private RuleLevel ruleLevel;

	private int amount;

	/*CONSTRUCTOR*/
	public LevelReward() {	}



	public Attribute getAttribute() {
		return attribute;
	}

	public void setAttribute(Attribute attribute) {
		this.attribute = attribute;
	}

	public RuleLevel getRuleLevel() {
		return ruleLevel;
	}

	public void setRuleLevel(RuleLevel ruleLevel) {
		this.ruleLevel = ruleLevel;
	}

	public int getAmount() {
		return amount;
	}

	public void setAmount(int amount) {
		this.amount = amount;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
}
```

LogEvent.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import java.sql.Date;
import java.sql.Timestamp;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class LogEvent {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

//	@ManyToOne
//	private Rule rule;
//
//	@ManyToOne
//	private User user;

	@Column(name="rule_id")
	private int rule;

	@Column(name="user_id")
	private int user;

	@Column(name="event_date")
	private Timestamp dateEvent;

	public LogEvent() {
```

```
            // TODO Auto-generated constructor stub
      }

      public int getId() {
            return id;
      }

      public void setId(int id) {
            this.id = id;
      }

//    public Rule getRule() {
//          return rule;
//    }
//
//    public void setRule(Rule rule) {
//          this.rule = rule;
//    }
//
//    public User getUser() {
//          return user;
//    }
//
//    public void setUser(User user) {
//          this.user = user;
//    }



      public Timestamp getDateEvent() {
            return dateEvent;
      }

      public int getRule() {
            return rule;
      }

      public void setRule(int rule) {
            this.rule = rule;
      }
```

```
	public int getUser() {
		return user;
	}

	public void setUser(int user) {
		this.user = user;
	}

	public void setDateEvent(Timestamp dateEvent) {
		this.dateEvent = dateEvent;
	}
}
```

Rule.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;

@Entity
public class Rule {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	@NotNull
	@Column(unique=true, nullable=false)
	private String name;

	@NotNull
	private String type;

	@NotNull
	@Column(name="times_to_complete")
	private int timesToComplete;
```

```
@NotNull
@Column(name="xp_reward")
private int xp;

@NotNull
private boolean active;

@NotNull
private boolean repeatable;

@NotNull
private boolean finished;

public Rule() {

}

public int getId() {
	return id;
}

public void setId(int id) {
	this.id = id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public String getType() {
	return type;
}

public void setType(String type) {
	this.type = type;
}
```

```
public boolean isActive() {
      return active;
}

public void setActive(boolean active) {
      this.active = active;
}


public int getTimesToComplete() {
      return timesToComplete;
}

public void setTimesToComplete(int timesToComplete) {
      this.timesToComplete = timesToComplete;
}

public int getXp() {
      return xp;
}

public void setXp(int xp) {
      this.xp = xp;
}

public boolean isFinished() {
      return finished;
}

public void setFinished(boolean finished) {
      this.finished = finished;
}

public boolean isRepeatable() {
      return repeatable;
}

public void setRepeatable(boolean repeatable) {
      this.repeatable = repeatable;
}
```

```
}
```

RuleAttribute.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.validation.constraints.NotNull;

@Entity
public class RuleAttribute {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	@ManyToOne
	private Rule rule;

	@ManyToOne
	private Attribute attribute;

	@NotNull
	private int amount;

	public long getId() {
		return id;
	}

	public Attribute getAttribute() {
		return attribute;
	}

	public void setAttribute(Attribute attribute) {
		this.attribute = attribute;
	}
```

```
	public void setId(int id) {
		this.id = id;
	}

	public Rule getRule() {
		return rule;
	}

	public void setRule(Rule rule) {
		this.rule = rule;
	}

	public int getAmount() {
		return amount;
	}

	public void setAmount(int amount) {
		this.amount = amount;
	}
}
```

RuleBadge.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;
import javax.validation.constraints.NotNull;

@Entity
public class RuleBadge {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;
```

```
    @OneToOne
    private Rule rule;

    @OneToOne
    private Badge badge;

    @ManyToOne
    private Attribute attribute;

    @Column(name="goal_value", nullable = true)
    private Integer goalValue;

    public long getId() {
        return id;
    }

    public Badge getBadge() {
        return badge;
    }

    public void setBadge(Badge badge) {
        this.badge = badge;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Rule getRule() {
        return rule;
    }

    public void setRule(Rule rule) {
        this.rule = rule;
    }

    public Attribute getAttribute() {
        return attribute;
    }

    public void setAttribute(Attribute attribute) {
```

```
            this.attribute = attribute;
      }

      public Integer getGoalValue() {
            return goalValue;
      }

      public void setGoalValue(Integer goalValue) {
            this.goalValue = goalValue;
      }

}
```

RuleBadgeAttribute.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

@Entity
@Table(name="rule_badge_attribute")
public class RuleBadgeAttribute {
      @Id
      @GeneratedValue(strategy = GenerationType.AUTO)
      private int id;

      @ManyToOne
      private Rule rule;

      @ManyToOne
      private Badge badge;

      @ManyToOne
      private Attribute attribute;
```

```
@NotNull
private int value;

public RuleBadgeAttribute() {
	// TODO Auto-generated constructor stub
}

public int getId() {
	return id;
}

public void setId(int id) {
	this.id = id;
}

public Rule getRule() {
	return rule;
}

public void setRule(Rule rule) {
	this.rule = rule;
}

public Badge getBadge() {
	return badge;
}

public void setBadge(Badge badge) {
	this.badge = badge;
}

public Attribute getAttribute() {
	return attribute;
}

public void setAttribute(Attribute attribute) {
	this.attribute = attribute;
}

public int getValue() {
```

```
            return value;
      }

      public void setValue(int value) {
            this.value = value;
      }
}
```

RuleLevel.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class RuleLevel {
      @Id
      @GeneratedValue(strategy = GenerationType.AUTO)
      private int id;

      @OneToOne
      private Rule rule;

      @Column(name="start_level")
      private int startLevel;

      @Column(name="end_level")
      private int endLevel;

      @Column(name="xp_to_level_up")
      private int xpToLevelUp;

      public RuleLevel() {
            // TODO Auto-generated constructor stub
      }
```

```
	public int getId() {
		return id;
	}

	public int getStartLevel() {
		return startLevel;
	}

	public void setStartLevel(int startLevel) {
		this.startLevel = startLevel;
	}

	public int getEndLevel() {
		return endLevel;
	}

	public void setEndLevel(int endLevel) {
		this.endLevel = endLevel;
	}

	public int getXpToLevelUp() {
		return xpToLevelUp;
	}

	public void setXpToLevelUp(int xpToLevelUp) {
		this.xpToLevelUp = xpToLevelUp;
	}

	public void setId(int id) {
		this.id = id;
	}

	public Rule getRule() {
		return rule;
	}

	public void setRule(Rule rule) {
		this.rule = rule;
	}
}
```

User.Java

---

package br.ufsc.tcc.gamifyEngine.model;

```
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToMany;
import javax.validation.constraints.NotNull;

import com.fasterxml.jackson.annotation.JsonManagedReference;

@Entity
public class User {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	private int xp;

	@Column(name="current_xp")
	private int currentXp;

	private int level;

	@NotNull
	private boolean active;
//	@OneToMany(cascade = CascadeType.ALL, mappedBy = "user",
orphanRemoval = true)
	@OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
	@JsonManagedReference (value="userAttributeReference")
```

```
	private List <UserAttribute> attributes;

	@JoinTable(name="user_badges", joinColumns = @JoinColumn(name
= "user_id", referencedColumnName="id"), inverseJoinColumns =
@JoinColumn(name = "badge_id", referencedColumnName="id" ))
	@ManyToMany
	private List <Badge> badges;

	@Column(nullable = true)
	private Integer extern_id;

	@Column(nullable = true)
	private String name;

	public User() {

	}

	public List<Badge> getBadges() {
		return badges;
	}

	public void setBadges(List<Badge> badges) {
		this.badges = badges;
	}

	public List<UserAttribute> getAttributes() {
		return attributes;
	}

	public void setAttributes(List<UserAttribute> attributes) {
		this.attributes = attributes;
	}

	public boolean isActive() {
		return active;
	}

	public void setActive(boolean active) {
		this.active = active;
	}
```

```
public int getLevel() {
	return level;
}

public void setLevel(int level) {
	this.level = level;
}

public int getId() {
	return id;
}

public void setId(int id) {
	this.id = id;
}

public int getXp() {
	return xp;
}

public void setXp(int xp) {
	this.xp = xp;
}

public int getCurrentXp() {
	return currentXp;
}

public void setCurrentXp(int currentXp) {
	this.currentXp = currentXp;
}

public Integer getExtern_id() {
	return extern_id;
}

public void setExtern_id(Integer extern_id) {
	this.extern_id = extern_id;
}
```

```
	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
}
```

UserAttribute.Java

```
package br.ufsc.tcc.gamifyEngine.model;

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.ManyToOne;
import javax.persistence.MapsId;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonBackReference;

import br.ufsc.tcc.gamifyEngine.compositeKeys.RuleAttributeKey;

@Entity
@Table(name="user_attributes")
public class UserAttribute {

	/*ATTRIBUTES*/
	@EmbeddedId
//	@GeneratedValue(strategy = GenerationType.AUTO)
	@JsonBackReference
	private RuleAttributeKey id;

	@MapsId("user")
	@ManyToOne
	@JsonBackReference(value="userAttributeReference")
	private User user;
```

```
	@MapsId("attribute")
	@ManyToOne
	private Attribute attribute;
	private int value;

	/*CONSTRUCTOR*/
	public UserAttribute() {	}


	/*GETTERS AND SETTERS*/
	public User getUser() {
		return user;
	}
	public void setUser(User user) {
		this.user = user;
	}
	public Attribute getAttribute() {
		return attribute;
	}
	public void setAttribute(Attribute attribute) {
		this.attribute = attribute;
	}
	public int getValue() {
		return value;
	}
	public void setValue(int value) {
		this.value = value;
	}
	public RuleAttributeKey getId() {
		return id;
	}
	public void setId(RuleAttributeKey id) {
		this.id = id;
	}
}
```

AttributeService.Java

```
package br.ufsc.tcc.gamifyEngine.service;
```

```
import java.util.List;

import br.ufsc.tcc.gamifyEngine.model.Attribute;

public interface AttributeService {
	public Attribute getAttribute(int attributeId);

	public void deleteAttribute(int attributeId);

	public Attribute saveAttribute(Attribute attribute);

	public Iterable<Attribute> findAllAttributes();
}
```

AttributeServiceImpl.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import br.ufsc.tcc.gamifyEngine.dao.AttributeDao;
import br.ufsc.tcc.gamifyEngine.model.Attribute;

@Service
public class AttributeServiceImpl implements AttributeService{
	@Autowired
	private AttributeDao attributeDao;

	@Override
	public Attribute getAttribute(int attributeId) {
		return attributeDao.findById(attributeId);
	}

	@Override
```

```
	public void deleteAttribute(int attributeId) {
		this.attributeDao.delete(attributeId);
	}

	@Override
	public Attribute saveAttribute(Attribute attribute) {
		return this.attributeDao.save(attribute);
	}

	@Override
	public Iterable<Attribute> findAllAttributes() {
		return this.attributeDao.findAll();
	}
}
```

BadgeService.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import br.ufsc.tcc.gamifyEngine.model.Badge;

public interface BadgeService {
	public Badge getBadge(int badgeId);

	public void deleteBadge(int badgeId);

	public Badge saveBadge(Badge badge);

	public Iterable<Badge> findAllbadges();
}
```

BadgeServiceImpl.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import br.ufsc.tcc.gamifyEngine.dao.BadgeDao;
import br.ufsc.tcc.gamifyEngine.model.Badge;

@Service
public class BadgeServiceImpl implements BadgeService{
	@Autowired
	private BadgeDao badgeDao;

	@Override
	public Badge getBadge(int badgeId) {
		return badgeDao.findById(badgeId);
	}

	@Override
	public void deleteBadge(int badgeId) {
		this.badgeDao.delete(badgeId);
	}

	@Override
	public Badge saveBadge(Badge badge) {
		return this.badgeDao.save(badge);
	}

	@Override
	public Iterable<Badge> findAllbadges() {
		return this.badgeDao.findAll();
	}
}
```

LogService.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import java.util.List;

import br.ufsc.tcc.gamifyEngine.model.LogEvent;

public interface LogService {
	public List<LogEvent> getLogByUserAndRule(int userId, int
ruleId);
```

```
	public void insertLog(LogEvent logEvent);

	public void deleteLog(int logId);

	public LogEvent getLog(int logId);

	public List<LogEvent> getUserLogs(int userId);

	public LogEvent saveLog(LogEvent logEvent);

	public void deleteRuleLogsFromUser(int ruleId, int userId);
}
```

LogServiceImpl.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import br.ufsc.tcc.gamifyEngine.dao.RuleDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleLevelDao;
import br.ufsc.tcc.gamifyEngine.dao.LogEventDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleAttributeDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleBadgeDao;
import br.ufsc.tcc.gamifyEngine.model.LogEvent;
import br.ufsc.tcc.gamifyEngine.model.Rule;
import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleBadge;
import br.ufsc.tcc.gamifyEngine.model.RuleLevel;

@Service
public class LogServiceImpl implements LogService{

	@Autowired
	private LogEventDao logEventDao;

	public LogServiceImpl() {
```

```
	}

	@Override
	public List<LogEvent>getLogByUserAndRule(int userId, int
ruleId) {
		return this.logEventDao.findByUserAndRule(userId,
ruleId);
	}

	@Override
	public void insertLog(LogEvent logEvent) {
		this.logEventDao.save(logEvent);

	}

	@Override
	public void deleteLog(int logId) {
		this.logEventDao.delete(logId);

	}

	@Override
	public LogEvent getLog(int logId) {
		return this.logEventDao.findById(logId);

	}

	@Override
	public List<LogEvent> getUserLogs(int userId) {
		return this.logEventDao.getUserLogs(userId);

	}

	@Override
	public LogEvent saveLog(LogEvent logEvent) {
		return this.logEventDao.save(logEvent);

	}

	@Override
```

```
	public void deleteRuleLogsFromUser(int ruleId, int userId) {
		this.logEventDao.deleteRuleLogsFromUser(ruleId,
userId);

	}
}
```

RuleService.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import java.util.List;
import java.util.Map;

import br.ufsc.tcc.gamifyEngine.model.Attribute;
import br.ufsc.tcc.gamifyEngine.model.LevelReward;
import br.ufsc.tcc.gamifyEngine.model.Rule;
import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleBadge;
import br.ufsc.tcc.gamifyEngine.model.RuleBadgeAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleLevel;
import br.ufsc.tcc.gamifyEngine.model.User;

public interface RuleService {
	public Rule getRule(int ruleId);
	public Iterable<Rule> findAllRules();
	public RuleAttribute getRuleAttribute(int ruleAttributeId);
	public RuleBadge getRuleBadge(int ruleBadgeId);
	public List<RuleAttribute> getRuleAttributeByRule(int rule);
	public RuleLevel getRuleLevel(int ruleLevelId);
	public RuleLevel findAdequatedRuleLevel(User user);
	public RuleBadge getRuleBadgesByRule(int ruleId);
	void deleteRuleLevel(int ruleLevelId);
	void deleteRuleBadge(int ruleBadgeId);
	void deleteRuleAttribute(int ruleAttributeId);
	void deleteRule(int ruleId);
	public Rule saveRule(Rule currentRule);
	public RuleBadge saveRuleBadge(RuleBadge ruleBadge);
	public RuleLevel saveRuleLevel(RuleLevel ruleLevel);
	public RuleAttribute saveRuleAttribute(RuleAttribute
ruleAttribute);
```

```
	public LevelReward saveLevelReward(LevelReward levelReward);
//	public LevelReward getLevelReward(int levelRewardId);
	public RuleBadgeAttribute
saveRuleBadgeAttribute(RuleBadgeAttribute ruleBadgeAttribute);
	public Iterable<RuleAttribute> findAllAttributeRules();
	public Iterable<RuleBadge> findAllBadgeRules();
	public Iterable<RuleLevel> findAllLevelRules();

	/**
	 * Esta função verifica recursivamente, se após um usuário
receber uma recompensa,
	 * outras regras não poderão ser desencadeadas, gerando novas
recompensas ou
	 * ou simplesmente fazendo atribuições necessárias.
	 * @param type tipo da recompensa conquistada (xp, level,
attributo ou badge)
	 * @param user Usuário que recebeu a recompensa
	 * @param attribute. Pode ser null. Refere-se ao attributo
que foi modificado
	 *
	 **/
	public void evaluate(String type, User user, Attribute
attribute);

	public Map<String, Object> processRule(int userId, int
ruleId);
	public Iterable<LevelReward> getLevelRewardsByRuleLevel(int
idRuleLevel);
	public List<Rule> getRuleByType(String type);



}
```

RuleServiceImpl.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.HashMap;
```

```java
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import br.ufsc.tcc.gamifyEngine.dao.LevelRewardDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleAttributeDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleBadgeAttributeDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleBadgeDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleDao;
import br.ufsc.tcc.gamifyEngine.dao.RuleLevelDao;
import br.ufsc.tcc.gamifyEngine.dao.UserDao;
import br.ufsc.tcc.gamifyEngine.model.Attribute;
import br.ufsc.tcc.gamifyEngine.model.Badge;
import br.ufsc.tcc.gamifyEngine.model.FeedbackAttributes;
import br.ufsc.tcc.gamifyEngine.model.FeedbackLevel;
import br.ufsc.tcc.gamifyEngine.model.FeedbackXp;
import br.ufsc.tcc.gamifyEngine.model.LevelReward;
import br.ufsc.tcc.gamifyEngine.model.LogEvent;
import br.ufsc.tcc.gamifyEngine.model.Rule;
import br.ufsc.tcc.gamifyEngine.model.RuleAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleBadge;
import br.ufsc.tcc.gamifyEngine.model.RuleBadgeAttribute;
import br.ufsc.tcc.gamifyEngine.model.RuleLevel;
import br.ufsc.tcc.gamifyEngine.model.User;
import br.ufsc.tcc.gamifyEngine.model.UserAttribute;

@Service
public class RuleServiceImpl implements RuleService{

	@Autowired
	private RuleDao ruleDao;

	@Autowired
	private UserDao userDao;

	@Autowired
	private LevelRewardDao levelRewardDao;

	@Autowired
```

```
    private RuleAttributeDao ruleAttributeDao;

    @Autowired
    private RuleBadgeDao ruleBadgeDao;

    @Autowired
    private RuleLevelDao ruleLevelDao;

    @Autowired
    private RuleBadgeAttributeDao ruleBadgeAttributeDao;

    @Autowired
    UserService userService;

    @Autowired
    LogService logService;

    List<FeedbackAttributes> attributesChanges;
    List<Badge> badgesChanges;
    FeedbackXp xpChanges;
    FeedbackLevel levelChanges;

    public RuleServiceImpl() {

    }

    @Override
    public Rule getRule(int ruleId) {
        return ruleDao.findById(ruleId);
    }

    @Override
    public Iterable<Rule> findAllRules() {
        return ruleDao.findAll();
    }

    @Override
    public RuleAttribute getRuleAttribute(int ruleAttributeId) {
        return ruleAttributeDao.findById(ruleAttributeId);
    }
```

```
	@Override
	public RuleBadge getRuleBadge(int ruleBadgeId) {
		return ruleBadgeDao.findById(ruleBadgeId);
	}

	@Override
	public List<RuleAttribute> getRuleAttributeByRule(int ruleId)
{
		return
ruleAttributeDao.findRuleAttributeByRuleId(ruleId);
	}


	@Override
	public RuleBadge getRuleBadgesByRule(int ruleId) {
		return this.ruleBadgeDao.findRuleBadgeByRuleId(ruleId);

	}

	@Override
	public RuleLevel getRuleLevel(int ruleLevelId) {
		return ruleLevelDao.findById(ruleLevelId);
	}


	@Override
	public void evaluate(String type, User user, Attribute att) {
//TODO mudar pra enum

		RuleLevel ruleLevel = null;

		switch (type) {
			case "xp":


				ruleLevel =
this.findAdequatedRuleLevel(user);

				int currentUserXp = user.getCurrentXp();

				if(ruleLevel != null) {
```

```
                            while(currentUserXp >=
ruleLevel.getXpToLevelUp()) {
                                  user.setLevel(user.getLevel() +
1);

this.levelChanges.setLevelIncreased(true);

this.levelChanges.setNewLevel(user.getLevel());


user.setCurrentXp(user.getCurrentXp() -
ruleLevel.getXpToLevelUp());
                                  currentUserXp =
user.getCurrentXp();
                                  this.evaluate("level", user,
null);
                                  ruleLevel =
this.findAdequatedRuleLevel(user);
                            }
                      }
                      break;
                case "badge":
                      //TODO asdasd
                      break;
                case "attribute":
                      //List<RuleBadgeAttribute> ruleBadgesAtt =
this.ruleBadgeAttributeDao.findByAttributeId(att.getId());
                      List<RuleBadge> ruleBadgesAtt =
this.ruleBadgeDao.findByAttributeId(att.getId());

                      for(RuleBadge ruleBadgeAtt: ruleBadgesAtt){


if(!user.getBadges().contains(ruleBadgeAtt.getBadge())) {
                                  for(UserAttribute userAttribute:
user.getAttributes()) {

if(userAttribute.getAttribute().getId() ==
ruleBadgeAtt.getAttribute().getId() && userAttribute.getValue() >=
ruleBadgeAtt.getGoalValue() )
                                        {
```

```
                                        //TODO verificar se
está no LOG

user.getBadges().add(ruleBadgeAtt.getBadge());

badgesChanges.add(ruleBadgeAtt.getBadge());
//
ruleBadgeAtt.getRule().setFinished(true);

this.ruleBadgeDao.save(ruleBadgeAtt);
                                    }
                                }
                            }
                        }
                        break;
                case "level":
                        ruleLevel =
this.findAdequatedRuleLevel(user);
                        List<LevelReward> levelRewards =
this.levelRewardDao.findCurrentLevelReward(ruleLevel.getId());

                        //atualiza os atributos do usuário
                        for (LevelReward levelReward : levelRewards)
{
                            int amount = levelReward.getAmount();
                            int attId =
levelReward.getAttribute().getId();

                            user.getAttributes().stream()
                                    .filter(attribute ->
attribute.getAttribute().getId() == attId)
                                    .forEach(attribute -> {

attribute.setValue(attribute.getValue() + amount);

                                        boolean hasAttribute =
false;

                                        for(FeedbackAttributes
feedbackAtt : this.attributesChanges) {
```

```
if(feedbackAtt.getAttribute().getId() ==
attribute.getAttribute().getId()) {
                                                        hasAttribute
= true;

feedbackAtt.setAmountChanged(feedbackAtt.getAmountChanged() +
amount);
                                                    }
                                                }

                                                if(!hasAttribute) {

this.attributesChanges.add(new
FeedbackAttributes(attribute.getAttribute(), amount));
                                                }
                                            });

                                this.evaluate("attribute", user,
levelReward.getAttribute());
                            }
                            break;
                        default:
                            break;
                    }
                }

                @Override
                public RuleLevel findAdequatedRuleLevel(User user) {
                    RuleLevel ruleLevel = null;
                    Integer highestLevelRule =
this.ruleLevelDao.findHighestLevelRange();

                    if(highestLevelRule > 0) {
                        if(highestLevelRule < user.getLevel())
                            ruleLevel =
this.ruleLevelDao.findByLevelRange(highestLevelRule);
                        else
                            ruleLevel =
this.ruleLevelDao.findByLevelRange(user.getLevel());

                        return ruleLevel;
```

```
        }
        return null;
    }

    @Override
    public void deleteRule(int ruleId) {
        this.ruleDao.delete(ruleId);
    }

    @Override
    public void deleteRuleAttribute(int ruleAttributeId) {
        this.ruleAttributeDao.delete(ruleAttributeId);
    }

    @Override
    public void deleteRuleBadge(int ruleBadgeId) {
        this.ruleBadgeDao.delete(ruleBadgeId);

    }

    @Override
    public void deleteRuleLevel(int ruleLevelId) {
        this.ruleLevelDao.delete(ruleLevelId);
    }

    @Override
    public Rule saveRule(Rule currentRule) {
        return this.ruleDao.save(currentRule);
    }

    @Override
    public RuleBadge saveRuleBadge(RuleBadge ruleBadge) {
        return this.ruleBadgeDao.save(ruleBadge);

    }

    @Override
    public RuleLevel saveRuleLevel(RuleLevel ruleLevel) {
        return this.ruleLevelDao.save(ruleLevel);
    }
```

```
	@Override
	public RuleAttribute saveRuleAttribute(RuleAttribute
ruleAttribute) {
		return this.ruleAttributeDao.save(ruleAttribute);
	}

	@Override
	public LevelReward saveLevelReward(LevelReward levelReward) {
		return this.levelRewardDao.save(levelReward);
	}

	@Override
	public RuleBadgeAttribute
saveRuleBadgeAttribute(RuleBadgeAttribute ruleBadgeAttribute) {
		return
this.ruleBadgeAttributeDao.save(ruleBadgeAttribute);
	}

	@Override
	public Iterable<RuleAttribute> findAllAttributeRules() {
		return this.ruleAttributeDao.findAll();
	}

	@Override
	public Iterable<RuleBadge> findAllBadgeRules() {
		return this.ruleBadgeDao.findAll();
	}

	@Override
	public Iterable<RuleLevel> findAllLevelRules() {
		return this.ruleLevelDao.findAll();
	}

	@Override
	public Iterable<LevelReward> getLevelRewardsByRuleLevel(int
idRuleLevel) {
		return
this.levelRewardDao.getLevelRewardsByRuleLevel(idRuleLevel);
	}
```

```
	@Override
	public Map<String, Object> processRule(int userId, int
ruleId) {

		Map< String, Object > feedBackChanges = new HashMap<
String, Object>();
		this.attributesChanges = new
ArrayList<FeedbackAttributes>();
		this.badgesChanges = new ArrayList<Badge>();
		this.xpChanges = new FeedbackXp();
		this.levelChanges = new FeedbackLevel();

		Rule rule = this.getRule(ruleId);
		User user = userService.getUser(userId);
		String ruleType = rule.getType();

		boolean completed = false;

		List<LogEvent> logs =
this.logService.getLogByUserAndRule(userId, ruleId);
		int timesCompleted = logs.size();

		if(rule.getTimesToComplete() <= timesCompleted ) {
			if(!rule.isRepeatable())
				completed = true;
		}

		User userUpdated = null;

		if(!completed) {

			timesCompleted++;

			//TODO ruleType troca pra enum depois
			switch (ruleType) {
			case "attribute":
				List<RuleAttribute> ruleAttributes = null;
				ruleAttributes =
this.getRuleAttributeByRule(ruleId);

				//atualiza os atributos do usuário
```

```
                    for (RuleAttribute ruleAttribute :
ruleAttributes) {
                         int amount = ruleAttribute.getAmount();
                         int attId =
ruleAttribute.getAttribute().getId();

                         user.getAttributes().stream()
                                   .filter(attribute ->
attribute.getAttribute().getId() == attId)
                                   .forEach( attribute -> {

attribute.setValue(attribute.getValue() + amount);

this.evaluate("attribute", user, attribute.getAttribute());

attributesChanges.add(new
FeedbackAttributes(attribute.getAttribute(), amount));
                                   });
                    }
                    break;
               case "badge":
                    RuleBadge ruleBadge = null;
                    ruleBadge =
this.getRuleBadgesByRule(ruleId);
                    if(timesCompleted >=
rule.getTimesToComplete()) {

user.getBadges().add(ruleBadge.getBadge());

badgesChanges.add(ruleBadge.getBadge());
                         this.evaluate("badge", user, null);
                    }
                    break;
               default:
                    break;
               }

               //sempre vai ganhar XP, mesmo que a rule não for
totalmente completa
               if(rule.getXp() != 0) {
                    user.setXp(user.getXp() + rule.getXp());
```

```
                    user.setCurrentXp(user.getCurrentXp() +
rule.getXp());
                    this.evaluate("xp", user, null);

                    xpChanges.setXpChanged(true);
                    xpChanges.setAmountChanged(rule.getXp());
                    xpChanges.setUserTotalXp(user.getXp());

xpChanges.setUserNewCurrentXp(user.getCurrentXp());



               }

               userUpdated = userService.saveUser(user);

               //faz log do evento
               LogEvent logEvent = new LogEvent();
               logEvent.setRule(rule.getId());
               logEvent.setUser(user.getId());
               Timestamp timestamp = new
Timestamp(System.currentTimeMillis());
               logEvent.setDateEvent(timestamp);
               this.logService.insertLog(logEvent);
          }


          Map<String, Object> response = new HashMap<String,
Object>();

          feedBackChanges.put("attributes", attributesChanges);
          feedBackChanges.put("badges", badgesChanges);
          feedBackChanges.put("experience", xpChanges);
          feedBackChanges.put("level", levelChanges);

          if(userUpdated != null)
               response.put("user", userUpdated);
          else
               response.put("user", user);

          response.put("feedBackChanges", feedBackChanges);
```

```
            return response;
      }

      @Override
      public List<Rule> getRuleByType(String type) {
            return this.ruleDao.findRuleByType(type);
      }

//    @Override
//    public LevelReward getLevelReward(int levelRewardId) {
//          return levelRewardDao.findById(levelRewardId);
//
//    }
}
```

UserService.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import br.ufsc.tcc.gamifyEngine.model.User;

public interface UserService {
      public User getUser(int userId);
      public Iterable<User> findAllUsers();
      public User saveUser(User user);
      public void deleteUser(int userId);
      public void getUserBadge(int userId, int badgeId);
      public void deleteUserBadge(int userId, int badgeId);
}
```

User ServiceImpl.Java

```
package br.ufsc.tcc.gamifyEngine.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```java
import br.ufsc.tcc.gamifyEngine.dao.UserDao;
import br.ufsc.tcc.gamifyEngine.model.User;

@Service
public class UserServiceImpl implements UserService {
	@Autowired
	private UserDao userDao;

	public UserServiceImpl() {

	}

	@Override
	public User getUser(int userId) {
		return userDao.findById(userId);
	}

	@Override
	public Iterable<User> findAllUsers() {
		return userDao.findAll();
	}

	@Override
	public User saveUser(User user) {
		return userDao.save(user);
	}

	@Override
	public void deleteUser(int userId) {
		this.userDao.delete(userId);
	}

	@Override
	public void getUserBadge(int userId, int badgeId) {
		// TODO Auto-generated method stub

	}

	@Override
	public void deleteUserBadge(int userId, int badgeId) {
		this.userDao.deleteUserBadge(userId, badgeId);
```

```
      }
}
```

style.css

```
body {
      overflow-y: scroll;
}

.container {
      margin-bottom:50px;
}
```

app.js

```
var app = angular.module('app',
['ngRoute','ngResource','ngTable']);
app.config(function($routeProvider){
    $routeProvider
        .when('/usuarios',{
            templateUrl: '/views/user/users.html',
            controller: 'usersController'
        })
        .when('/usuarios/novo',{
             templateUrl: '/views/user/new-user.html',
             controller: 'newUserController'
         })
         .when('/usuarios/:idUser',{
            templateUrl: '/views/user/new-user.html',
            controller: 'editUserController'
        })
        .when('/insignias',{
            templateUrl: '/views/badge/badges.html',
            controller: 'badgesController'
        })
        .when('/insignias/novo',{
            templateUrl: '/views/badge/new-badge.html',
            controller: 'newBadgeController'
```

```
        })
        .when('/insignias/:idBadge',{
            templateUrl: '/views/badge/new-badge.html',
            controller: 'editBadgeController'
        })
        .when('/atributos',{
            templateUrl: '/views/attribute/attributes.html',
            controller: 'attributesController'
        })
        .when('/atributos/novo',{
            templateUrl: '/views/attribute/new-attribute.html',
            controller: 'newAttributeController'
        })
        .when('/atributos/:idAttribute',{
            templateUrl: '/views/attribute/new-attribute.html',
            controller: 'editAttributeController'
        })
        .when('/regras-atributo',{
            templateUrl:
'/views/ruleAttribute/rules-attribute.html',
            controller: 'ruleAttributeController'
        })
        .when('/regras-atributo/novo',{
            templateUrl:
'/views/ruleAttribute/new-rule-attribute.html',
            controller: 'newRuleAttributeController'
        })
        .when('/regras-atributo/:idRule',{
            templateUrl:
'/views/ruleAttribute/new-rule-attribute.html',
            controller: 'newRuleAttributeController'
        })
        .when('/regras-insignia',{
            templateUrl: '/views/ruleBadge/rules-badge.html',
            controller: 'ruleBadgeController'
        })
        .when('/regras-insignia/novo',{
            templateUrl: '/views/ruleBadge/new-rule-badge.html',
            controller: 'newRuleBadgeController'
        })
        .when('/regras-insignia/:idRule',{
```

```
            templateUrl: '/views/ruleBadge/new-rule-badge.html',
            controller: 'newRuleBadgeController'
        })
        .when('/regras-nivel',{
            templateUrl: '/views/ruleLevel/rules-level.html',
            controller: 'ruleLevelController'
        })
        .when('/regras-nivel/novo',{
            templateUrl: '/views/ruleLevel/new-rule-level.html',
            controller: 'newRuleLevelController'
        })
        .when('/regras-nivel/:idRuleLevel',{
            templateUrl: '/views/ruleLevel/new-rule-level.html',
            controller: 'newRuleLevelController'
        })
        .when('/regras-xp',{
            templateUrl: '/views/ruleXp/rules-xp.html',
            controller: 'ruleXpController'
        })
        .when('/regras-xp/novo',{
            templateUrl: '/views/ruleXp/new-rule-xp.html',
            controller: 'newRuleXpController'
        })
        .when('/regras-xp/:idRule',{
            templateUrl: '/views/ruleXp/new-rule-xp.html',
            controller: 'newRuleXpController'
        })
        .otherwise(
            { redirectTo: '/'}
        );
});

app.run(['$rootScope','$timeout' , function ($rootScope, $timeout)
{
      $rootScope.showAlert = false;
      $rootScope.alertMessage = "";

      $rootScope.alert = function (msg) {

            $rootScope.showAlert = true;
            $rootScope.alertMessage = msg;
```

```
            $timeout(function() {
                $rootScope.showAlert = false;
            }, 3000);
        }
}]);

app.service('UserService', [ '$http', function($http) {

    this.getUser = function getUser(userId) {
        return $http({
            method : 'GET',
            url : 'api/users/' + userId
        });
    }

    this.addUser = function addUser(user) {
        return $http({
            method : 'POST',
            url : 'api/users/',
            data : user
        });
    }

    this.getAllUsers = function getAllUsers() {
        return $http({
            method : 'GET',
            url : 'api/users/'
        });
    }

    this.deleteUser = function deletelUser(userId) {
        return $http({
            method : 'DELETE',
            url : 'api/users/'+ userId
        });
    }


} ]);
```

```
app.service('BadgeService', [ '$http', function($http) {

     this.getAllBadges = function getAllBadge() {
        return $http({
            method : 'GET',
            url : 'api/badges/'
        });
    }

    this.getBadge = function getBadge(badgeId) {
        return $http({
            method : 'GET',
            url : 'api/badges/' + badgeId
        });
    }

    this.addBadge = function addBadge(badge) {
        return $http({
            method : 'POST',
            url : 'api/badges/',
            data : badge
        });
    }

     this.findAll = function findAll() {
        return $http({
            method : 'GET',
            url : 'api/badges/'
        });
    }

    this.deleteBadge = function deletelBadge(badgeId) {
        return $http({
            method : 'DELETE',
            url : 'api/badges/'+ badgeId
        });
    }

} ]);
```

```
app.service('AttributeService', [ '$http', function($http) {

     this.findAll = function findAll() {
        return $http({
            method : 'GET',
            url : 'api/attributes/'
        });
    }

    this.getAttribute = function getAttribute(attributeId) {
        return $http({
            method : 'GET',
            url : 'api/attributes/' + attributeId
        });
    }

    this.addAttribute = function addAttribute(attribute) {
        return $http({
            method : 'POST',
            url : 'api/attributes/',
            data : attribute
        });
    }

    this.deleteAttribute = function deleteAttribute(attributeId) {
        return $http({
            method : 'DELETE',
            url : 'api/attributes/'+ attributeId
        });
    }

} ]);

app.service('RuleService', [ '$http', function($http) {

    this.getRule = function getRule(ruleId) {
        return $http({
            method : 'GET',
            url : 'api/rules/' + ruleId
        });
```

```
    }

    this.getRuleBadge = function getRuleBadge(ruleBadgeId) {
        return $http({
            method : 'GET',
            url : 'api/badges/rules/' + ruleBadgeId
        });
    }

    this.getRuleAttributesByRule = function
getRuleAttributesByRule(ruleId) {
        return $http({
            method : 'GET',
            url : 'api/attributes/rules/byRule/' + ruleId
        });
    }

    this.addRule = function addRule(rule) {
        return $http({
            method : 'POST',
            url : 'api/rules/',
            data : rule
        });
    }
    this.addRuleAttribute = function
addRuleAttribute(ruleAttribute) {
     return $http({
           method : 'POST',
            url : 'api/attributes/rules/',
            data : ruleAttribute
        });
    }
    this.addRuleLevel = function addRuleLevel(ruleLevel) {
     return $http({
           method : 'POST',
            url : 'api/level/rules/',
            data : ruleLevel
        });
    }

    this.addRuleBadge = function addRuleBadge(ruleBadge) {
```

```
      return $http({
            method : 'POST',
             url : 'api/badges/rules/',
             data : ruleBadge
         });
    }

    this.addLevelReward = function addLevelReward(levelReward) {
      return $http({
            method : 'POST',
             url : 'api/level/rewards/',
             data : levelReward
         });
    }

    this.addRuleBadgeAttribute = function
addRuleBadgeAttribute(ruleBadgeAttribute) {
      return $http({
            method : 'POST',
             url : 'api/badges/attributes/rewards/',
             data : ruleBadgeAttribute
         });
    }


    this.getAllAttributeRules = function getAllAttributeRules() {
        return $http({
            method : 'GET',
            url : 'api/attributes/rules/'
        });
    }

    this.getAllBadgeRules = function getAllBadgeRules() {
        return $http({
            method : 'GET',
            url : 'api/badges/rules/'
        });
    }

    this.getAllBadgeRuleAttributes = function
getAllBadgeRuleAttributes() {
```

```
      return $http({
            method : 'GET',
             url : 'api/badges/attributes/'
        });
    }

    this.getAllLevelRules = function getAllLevelRules() {
        return $http({
            method : 'GET',
            url : 'api/level/rules/'
        });
    }

    this.deleteRule= function deleteRule(ruleId) {
        return $http({
            method : 'DELETE',
            url : 'api/rules/'+ ruleId
        });
    }

    this.deleteRuleAttribute = function
deleteRuleAttribute(ruleAttributeId) {
        return $http({
            method : 'DELETE',
            url : 'api/attributes/rules/'+ ruleAttributeId
        });
    }

    this.getRuleLevel = function getRuleLevel(idRuleLevel) {
        return $http({
            method : 'GET',
            url : 'api/level/rules/' + idRuleLevel
        });
    }

    this.getRuleLevelRewardsByRuleLevel = function
getRuleLevelRewardsByRuleLevel(idRuleLevel) {
        return $http({
            method : 'GET',
            url : 'api/level/rewards/rules/' + idRuleLevel
        });
```

```
    }

    this.findAllXpRules = function findAllXpRules(idRule) {
        return $http({
            method : 'GET',
            url : 'api/rules/type/'+ 'xp'
        });
    }

} ]);
```

attribute-controller.js

```
app.controller('attributesController', [ '$scope', '$rootScope',
'$timeout', 'AttributeService','NgTableParams', function($scope,
$rootScope, $timeout, AttributeService, NgTableParams) {

     $scope.attributesTableParams = new NgTableParams({}, {
          getData: function(params) {

               return AttributeService.findAll()
               .then (function success(response) {
                    $scope.listUsers = response.data;
                    params.total(response.data.length);
                    return response.data;

               },
               function error(response) {
                    $rootScope.alert('Error adding attribute!');
               });
          }
     });

     $scope.deleteAttribute = function(attributeId){
          AttributeService.deleteAttribute(attributeId)
          .then (function success(response) {
               $rootScope.alert('Atributo excluído com
sucesso!');
          },
          function error(response) {
```

```
                $rootScope.alert('Error adding attribute!');
            });
     }

}]);

app.controller('newAttributeController', [ '$scope', '$rootScope',
'$timeout', 'AttributeService', function($scope, $rootScope,
$timeout, AttributeService) {
    $scope.attribute = {};

    $scope.saveAttribute = function () {
     AttributeService.addAttribute($scope.attribute)
        .then (function success(response) {
           $rootScope.alert('Atributo adicionado com sucesso!');
        },
        function error(response) {
           $rootScope.alert('Error adding attribute!');
      });
    }
}]);

app.controller('editAttributeController', [ '$scope',
'$rootScope', '$timeout', '$routeParams', 'AttributeService',
function($scope, $rootScope, $timeout, $routeParams,
AttributeService) {
     $scope.attribute = {};

    AttributeService.getAttribute($routeParams.idAttribute)
    .then (function success(response) {
     $scope.attribute = response.data;
    },
    function error(response) {
     $rootScope.alert('Error adding attribute!');
    });

    $scope.saveAttribute = function () {
     AttributeService.addAttribute($scope.attribute)
        .then (function success(response) {
           $rootScope.alert('Atributo adicionado com sucesso!');
        },
```

```
        function error(response) {
           $rootScope.alert('Error adding attribute!');
       });
    }
}]);
```

badge-controller.js

```
app.controller('badgesController', [ '$scope', '$rootScope',
'$timeout', 'BadgeService','NgTableParams', function($scope,
$rootScope, $timeout, BadgeService, NgTableParams) {
     $scope.teste = "string";
     $scope.badgesTableParams = new NgTableParams({}, {
          getData: function(params) {

               return BadgeService.getAllBadges()
               .then (function success(response) {
                    $scope.listUsers = response.data;
                    params.total(response.data.length);
                    return response.data;

               },
               function error(response) {
                    $rootScope.alert('Error adding user!');
               });
          }
     });

     $scope.deleteBadge = function(badgeId){
          BadgeService.deleteBadge(badgeId)
          .then (function success(response) {
               $rootScope.alert('Insígnia excluída com
sucesso!');
          },
          function error(response) {
               $rootScope.alert('Error adding user!');
          });
     }

}]);
```

```
app.controller('newBadgeController', [ '$scope', '$rootScope',
'$timeout', 'BadgeService', function($scope, $rootScope, $timeout,
BadgeService) {
    $scope.badge = {};

    $scope.saveBadge = function () {
      BadgeService.addBadge($scope.badge)
         .then (function success(response) {
            $rootScope.alert('Emblema adicionado com sucesso!');
         },
         function error(response) {
            $rootScope.alert('Error adding user!');
       });
    }
}]);


app.controller('editBadgeController', [ '$scope', '$rootScope',
'$timeout', 'BadgeService', '$routeParams', function($scope,
$rootScope, $timeout, BadgeService, $routeParams) {
    $scope.badge = {};
    BadgeService.getBadge($routeParams.idBadge)
    .then (function success(response) {
      $scope.badge = response.data;
    },
    function error(response) {
      $rootScope.alert('Error adding user!');
    });


    $scope.saveBadge = function () {
      BadgeService.addBadge($scope.badge)
         .then (function success(response) {
            $rootScope.alert('Emblema adicionado com sucesso!');
         },
         function error(response) {
            $rootScope.alert('Error adding user!');
       });
    }
}]);
```

rule-badge-controller.js

```
app.controller('ruleBadgeController', [ '$scope', '$rootScope',
'$timeout', 'RuleService','NgTableParams', function($scope,
$rootScope, $timeout, RuleService, NgTableParams) {
	$scope.ruleBadgeTableParams = new NgTableParams({}, {
		getData: function(params) {
			return RuleService.getAllBadgeRules()
			.then (function success(response) {
				params.total(response.data.length);
				return response.data;
			},
			function error(response) {
				$rootScope.alert('Error adding user!');
			});
		}
	});

	$scope.deleteRule = function(ruleId){
		RuleService.deleteRule(ruleId)
		.then (function success(response) {
			$rootScope.alert('Regra de atributo excluída com
sucesso!');
		},
		function error(response) {
			$rootScope.alert('Error adding user!');
		});
	}
}]);


app.controller('newRuleBadgeController', [ '$scope', '$rootScope',
'$timeout', 'RuleService',
'BadgeService','AttributeService','$routeParams', function($scope,
$rootScope, $timeout, RuleService, BadgeService, AttributeService,
$routeParams) {
	$scope.rule = {type:"badge", finished:false, active:true,
repeatable:false, timesToComplete:1};
	$scope.ruleBadges = [{}];
```

```
      $scope.listBadges = [];
      $scope.listAttributes = [];
      $scope.selectedAttribute = {};

    AttributeService.findAll()
    .then (function success(response) {
      $scope.listAttributes = response.data;
    },
    function error(response) {
      $rootScope.alert('Error!');
    });

    BadgeService.findAll()
    .then (function success(response) {
      $scope.listBadges = response.data;
    },
    function error(response) {
      $rootScope.alert('Error!');
    });

    //EDIÇÃO
    if(typeof $routeParams.idRule != 'undefined') {
      console.log('asdasda');
      RuleService.getRuleBadge($routeParams.idRule)
         .then (function success(response) {
            console.log(response);
            console.log('asdasda');


            $scope.rule = response.data.rule;
            $scope.ruleBadges[0].attribute =
response.data.attribute
            $scope.ruleBadges[0].badge = response.data.badge;
            $scope.ruleBadges[0].goalValue =
response.data.goalValue;
            $scope.ruleBadges[0].id = response.data.id;
         },
         function error(response) {
            $rootScope.alert('Error!');
         });
```

```
    }

    $scope.saveRule = function () {
      RuleService.addRule($scope.rule)
      .then (function success(response)
      {
            $scope.ruleBadges.forEach(function(ruleBadge)
            {
                  ruleBadge.rule = response.data;
                  RuleService.addRuleBadge(ruleBadge)
                  .then(function success(response)
                  {
                        $rootScope.alert('Regra de insígnia
adicionada com sucesso!');
                  });
            });
        },
        function error(response) {
            $rootScope.alert('Error adding rule!');
       });
    }
}]);
```

rule-level-controller.js

```
app.controller('ruleLevelController', [ '$scope', '$rootScope',
'$timeout', 'RuleService','NgTableParams', function($scope,
$rootScope, $timeout, RuleService, NgTableParams) {

     $scope.ruleLevelTableParams = new NgTableParams({}, {
            getData: function(params) {
                  return RuleService.getAllLevelRules()
                  .then (function success(response) {
                        console.log(response);

                        params.total(response.data.length);
                        return response.data;
                  },
                  function error(response) {
                        $rootScope.alert('Error adding user!');
```

```
                });
            }
      });

      $scope.deleteRule = function(ruleId){
            RuleService.deleteRule(ruleId)
            .then (function success(response) {
                  $rootScope.alert('Regra excluída com sucesso!');
            },
            function error(response) {
                  $rootScope.alert('Error adding user!');
            });
      }

}]);


app.controller('newRuleLevelController', [ '$scope', '$rootScope',
'$timeout', 'RuleService','AttributeService', '$routeParams',
function($scope, $rootScope, $timeout, RuleService,
AttributeService, $routeParams) {
      $scope.rule = {type:'level', timesToComplete:0,
finished:false, active:true, repeatable:false, xp:0};
      $scope.ruleLevels = [{}];
      $scope.levelRewards = [{}];
      $scope.attributes = [];

    AttributeService.findAll()
    .then (function success(response) {
      $scope.attributes = response.data;
    },
    function error(response) {
      $rootScope.alert('Error!');
    });

    $scope.saveRule = function () {
      RuleService.addRule($scope.rule)
      .then (function success(response)
      {
            $scope.ruleLevels.forEach(function(ruleLevel)
            {
```

```
                ruleLevel.rule = response.data;
                RuleService.addRuleLevel(ruleLevel)
                .then(function success(response)
                {
                      alert('1');
                      console.log('AAAAAAAAAAAAAAAAAAAA');
                      console.log($scope.levelRewards);

$scope.levelRewards.forEach(function(levelReward)
                      {
                            console.log('bbbbbbbbbbbbbbbbbbbb');
                            levelReward.ruleLevel = response.data;

RuleService.addLevelReward(levelReward);
                      });
                });
           });

           $rootScope.alert('Regra de nível adicionada com
sucesso!');
        },
        function error(response)
        {
           $rootScope.alert('Error adding rule!');
      });
    }

    $scope.addLevelReward = function () {
      $scope.levelRewards.push({});
    }

    //EDIÇÃO
    if(typeof $routeParams.idRuleLevel != 'undefined') {

      console.log('asdasda');

      //get rule level
      RuleService.getRuleLevel($routeParams.idRuleLevel)
         .then (function success(response) {
            console.log(response);
            console.log('aaaaaaaaaaaaaaaaaaaaaaaaa');
```

```
            $scope.rule = response.data.rule;
            $scope.ruleLevels[0] = response.data;
        },
        function error(response) {
            $rootScope.alert('Error!');
        });

     //get rule level rewards

RuleService.getRuleLevelRewardsByRuleLevel($routeParams.idRuleLeve
l)
        .then (function success(response) {
            console.log(response);
            console.log('bbbbbbbbbbbbbbbbbbbb');
            if(response.data)
                  $scope.levelRewards = response.data;
        },
        function error(response) {
            $rootScope.alert('Error!');
        });
    }


}]);
```

rule-attribute-controller.js

```
app.controller('ruleAttributeController', [ '$scope',
'$rootScope', '$timeout', 'RuleService','NgTableParams',
function($scope, $rootScope, $timeout, RuleService, NgTableParams)
{
     $scope.ruleAttributeTableParams = new NgTableParams({}, {
          getData: function(params) {
               return RuleService.getAllAttributeRules()
               .then (function success(response) {
                    params.total(response.data.length);
                    return response.data;
               },
               function error(response) {
                    $rootScope.alert('Error adding user!');
```

```
                });
            }
        });

        $scope.deleteRuleAttribute = function(ruleAttributeId){
            RuleService.deleteRuleAttribute(ruleAttributeId)
            .then (function success(response) {
                $rootScope.alert('Regra de atributo excluída com
sucesso!');
            },
            function error(response) {
                $rootScope.alert('Error adding user!');
            });
        }

}]);

app.controller('newRuleAttributeController', [ '$scope',
'$rootScope', '$timeout', 'RuleService', 'AttributeService',
'$routeParams', function($scope, $rootScope, $timeout,
RuleService, AttributeService, $routeParams) {
    $scope.rule = {type:'attribute', finished:false, active:true,
repeatable:false};
    $scope.ruleAttributes = [{}];
    $scope.attributes = [];

    AttributeService.findAll()
    .then (function success(response) {
      $scope.attributes = response.data;
    },
    function error(response) {
      $rootScope.alert('Error!');
    });


    if(typeof $routeParams.idRule != 'undefined') {

      console.log('111111');
      RuleService.getRuleAttributesByRule($routeParams.idRule)
         .then (function success(response) {
            console.log(response);
```

```
            console.log('222222');

            //todos possuem a mesma rule, por isso pegar a de [0]
            $scope.rule = response.data[0].rule;
            $scope.ruleAttributes = response.data;
//
        },
        function error(response) {
            $rootScope.alert('Error!');
        });

    }

    $scope.saveRule = function () {
      RuleService.addRule($scope.rule)
      .then (function success(response) {
            console.log(response);

            $rootScope.alert('Regra adicionada com sucesso!');

            $scope.ruleAttributes.forEach(function(ruleAttribute){
                  ruleAttribute.rule = response.data;

                  RuleService.addRuleAttribute(ruleAttribute)
                  .then(function success(response){
                        $rootScope.alert('Regra de atributo
adicionada com sucesso!');
                  });
            });
        },
        function error(response) {
            $rootScope.alert('Error adding rule!');
       });
    }

    $scope.addAttribute = function () {
      $scope.ruleAttributes.push({});
    }
}]);
```

rule-xp-controller.js

```
app.controller('ruleXpController', [ '$scope', '$rootScope',
'$timeout', 'RuleService','NgTableParams', function($scope,
$rootScope, $timeout, RuleService, NgTableParams) {

	$scope.rulesTableParams = new NgTableParams({}, {
		getData: function(params) {
			return RuleService.findAllXpRules()
			.then (function success(response) {
				console.log(response);
				params.total(response.data.length);
				return response.data;

			},
			function error(response) {
				$rootScope.alert('Error adding attribute!');
			});
		}
	});

	$scope.deleteRule = function(ruleId){
		RuleService.deleteRule(ruleId)
		.then (function success(response) {
			$rootScope.alert('Regra excluída com sucesso!');
		},
		function error(response) {
			$rootScope.alert('Error adding attribute!');
		});
	}

}]);

app.controller('newRuleXpController', [ '$scope', '$rootScope',
'$timeout', 'RuleService','NgTableParams', '$routeParams',
function($scope, $rootScope, $timeout, RuleService, NgTableParams,
$routeParams) {

	$scope.rule = {type:'xp', finished:false, active:true,
repeatable:false};
```

```
    if(typeof $routeParams.idRule != 'undefined') {

          RuleService.getRule($routeParams.idRule)
         .then (function success(response) {
          console.log(response);
          $scope.rule = response.data;
         },
         function error(response) {
          $rootScope.alert('Error!');
         });
    }

    $scope.saveRule = function () {
          RuleService.addRule($scope.rule)
          .then (function success(response) {
          $rootScope.alert('Regra adicionada com sucesso!');
         },
         function error(response) {
          $rootScope.alert('Error adding rule!');
       });
    }
}]);
```

user-controller.js

```
app.controller('usersController', [ '$scope', '$rootScope',
'$timeout', 'UserService','NgTableParams', function($scope,
$rootScope, $timeout, UserService, NgTableParams) {

    $scope.usersTableParams = new NgTableParams({}, {
          getData: function(params) {

               return UserService.getAllUsers()
               .then (function success(response) {
                    $scope.listUsers = response.data;
                    params.total(response.data.length);
                    return response.data;
```

```
                },
                function error(response) {
                      $rootScope.alert('Error adding user!');
                });
           }
      });


      $scope.deleteUser = function(userId){
           UserService.deleteUser(userId)
           .then (function success(response) {
                $rootScope.alert('Usuário excluído com sucesso!');
           },
           function error(response) {
                $rootScope.alert('Error adding user!');
           });
      }
}]);

app.controller('newUserController', [ '$scope', '$rootScope',
'$timeout', 'UserService', function($scope, $rootScope, $timeout,
UserService,) {
    $scope.user = {level:1, current_xp:0, xp:0, active:true};

    $scope.saveUser = function () {
      UserService.addUser($scope.user)
         .then (function success(response) {
            $rootScope.alert('Usuário adicionado com sucesso!');
         },
         function error(response) {
            $rootScope.alert('Error adding user!');
       });
    }
}]);

app.controller('editUserController', [ '$scope', '$rootScope',
'$timeout', '$routeParams', 'UserService', function($scope,
$rootScope, $timeout, $routeParams, UserService) {
      $scope.user = {};

    UserService.getUser($routeParams.idUser)
```

```
    .then (function success(response) {
      $scope.user = response.data;
    },
    function error(response) {
      $rootScope.alert('Error adding user!');
    });

    $scope.saveUser = function () {
      UserService.addUser($scope.user)
         .then (function success(response) {
            $rootScope.alert('Usuário atualizado com sucesso!');
         },
         function error(response) {
            $rootScope.alert('Error adding user!');
       });
    }
}]);
```

attributes.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <a href="#/atributos/novo" type="button" class="btn
btn-primary pull-right">Novo attributo</a>
     <table ng-table="attributesTableParams" class="table"
show-filter="false">
          <tr ng-repeat="attribute in $data">
              <td title="'ID'" filter="" sortable="''">
                  {{attribute.id}}
              </td>

              <td title="'Nome'" filter="{ name: 'text'}"
sortable="''">
                  {{attribute.name}}
              </td>

              <td title="'Descrição'" filter="" sortable="''">
                  {{attribute.description}}
```

```
            </td>
            <td>
                    <div class="dropdown">
                        <button class="btn btn-default
dropdown-toggle" type="button"
                                id="dropdownMenu1"
data-toggle="dropdown" aria-haspopup="true"
                                aria-expanded="true">
                                opções <span
class="caret"></span>
                        </button>
                        <ul class="dropdown-menu"
aria-labelledby="dropdownMenu1">
                                <li><a
ng-click="deleteAttribute(attribute.id)" href="">Excluir</a></li>
                                <li><a ng-click=""
href="#/atributos/{{attribute.id}}">Editar</a></li>
                        </ul>
                    </div>
                </td>
        </tr>
    </table>
</div>
```

new-attribute.html

```
<div class="container">
    <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
    <h2>Novo atributo</h2>
    <form>
        <div class="form-group">
            <label for="attributeName">Nome</label>
            <input type="text" ng-model="attribute.name"
class="form-control" id="attributeName">
        </div>

        <div class="form-group">
```

```
                <label
for="attributeDescription">Descrição</label>
                <textarea type="text"
ng-model="attribute.description" class="form-control"
id="attributeDescription"></textarea>
            </div>

            <button type="submit" class="btn btn-success"
ng-click="saveAttribute()">Salvar</button>

      </form>
</div>
```

bagdes.html

```
<div class="container">
      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
      <a href="#/insignias/novo" type="button" class="btn
btn-primary pull-right">Nova insígnia</a>
      <table ng-table="badgesTableParams" class="table"
show-filter="false">
          <tr ng-repeat="badge in $data">

            <td title="'ID'" filter="" sortable="'id'">
                  {{badge.id}}
              </td>

              <td title="'Nome'" filter="{ name: 'text'}"
sortable="'name'">
                  {{badge.name}}
              </td>

              <td title="'Descrição'" filter="" sortable="'description'">
                  {{badge.description}}
              </td>

              <td>
                      <div class="dropdown">
                          <button class="btn btn-default
```

```
dropdown-toggle" type="button"
                                id="dropdownMenu1"
data-toggle="dropdown" aria-haspopup="true"
                                aria-expanded="true">
                                opções <span
class="caret"></span>
                            </button>
                            <ul class="dropdown-menu"
aria-labelledby="dropdownMenu1">
                                <li><a
ng-click="deleteBadge(badge.id)" href="">Excluir</a></li>
                                <li><a ng-click=""
href="#/insignias/{{badge.id}}">Editar</a></li>
                            </ul>
                        </div>
                </td>

            </tr>
        </table>
</div>
```

new-badge.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <h2>Nova insígnia</h2>
     <form>
          <div class="form-group">
               <label for="badgeName">Nome</label>
               <input type="text" ng-model="badge.name"
class="form-control" id="badgeName">
          </div>

          <div class="form-group">
               <label for="badgeDescription">Descrição</label>
               <textarea type="text" ng-model="badge.description"
class="form-control" id="badgeDescription" ></textarea>
          </div>
```

```
      </form>

      <button type="submit" class="btn btn-success"
ng-click="saveBadge()">Salvar</button>

</div>
```

new-rule-attribute.html

```
<div class="container">
      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
      <h2>Nova regra de atributo</h2>
      <form>
            <div class="form-group">
                  <label for="ruleName">Nome:</label>
                  <input type="text" ng-model="rule.name"
class="form-control" id="ruleName" placeholder="Nome">
            </div>

            <div class="form-group">
                  <label for="ruleTimeToComplete">Vezes para
completar:</label>
                  <input ng-model="rule.timesToComplete"
type="number" class="form-control" id="ruleTimeToComplete"
placeholder="Ex.: 10">
            </div>

            <div class="form-group">
                  <label for="ruleXpReward">Recompensa em
XP:</label>
                  <input ng-model="rule.xp" type="number"
class="form-control" id="ruleXpReward" placeholder="Ex.: 100">
            </div>

            <div class="checkbox">
                  <label>
                        <input ng-model="rule.active"
type="checkbox" ng-selected="true"> Ativar regra
```

```
                </label>
            </div>
            <div class="checkbox">
                <label>
                    <input ng-model="rule.repeatable"
type="checkbox" ng-selected="true"> Regra cíclica
                </label>
            </div>

            <div id="ruleAttributesContainer" ng-repeat="ra in
ruleAttributes">
                <div class="form-group">
                    <label for="ruleAttribute"> Atributo:
</label><br>
                   <select ng-options="attribute as
attribute.name for attribute in attributes track by attribute.id"
class="form-control" name="ruleAttribute" id="ruleAttribute"
ng-model="ruleAttributes[$index].attribute">
                     <option value="">Selecione um
atributo...</option>
                   </select>
                </div>
                <div class="form-group">
                    <label for="ruleAttributeAmount">Valor da
recompensa:</label>
                    <input
ng-model="ruleAttributes[$index].amount" type="number"
class="form-control" id="ruleAttributeAmount" placeholder="Ex.:
100">
                </div>
            </div>


            <button type="submit" class="btn btn-primary"
ng-click="addAttribute()">Adicionar atributo</button>

            <button type="submit" class="btn btn-success"
ng-click="saveRule()">Salvar</button>

        </form>
</div>
```

rule-attribute.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <a href="#/regras-atributo/novo" type="button" class="btn
btn-primary pull-right">Nova regra de atributo</a>
     <table ng-table="ruleAttributeTableParams" class="table"
show-filter="false">
          <tr ng-repeat="ruleAttribute in $data">
            <td title="'ID'" filter="" sortable="''">
                {{ruleAttribute.rule.id}}
             </td>

             <td title="'Regra'" filter="{ name: 'text'}"
sortable="''">
                {{ruleAttribute.rule.name}}
             </td>

             <td title="'Vezes para completar'" filter=""
sortable="''">
                {{ruleAttribute.rule.timesToComplete}}
             </td>

             <td title="'Recompensa em XP'" filter=""
sortable="''">
                {{ruleAttribute.rule.xp}}
             </td>

             <td title="'Regra cíclila'" filter="" sortable="''">
                {{ruleAttribute.rule.repeatable}}
             </td>

             <td title="'Finalizada'" filter="" sortable="''">
```

```
                {{ruleAttribute.rule.finished}}
            </td>

            <td title="'Ativa'" filter="" sortable="''">
                {{ruleAttribute.rule.active}}
            </td>
            <td title="'Atributo recompensado'" filter=""
sortable="''">
                {{ruleAttribute.attribute.name}}
            </td>
            <td title="'Quantidade'" filter="" sortable="''">
                {{ruleAttribute.amount}}
            </td>
            <td>
                    <div class="dropdown">
                        <button class="btn btn-default
dropdown-toggle" type="button"
                                id="dropdownMenu1"
data-toggle="dropdown" aria-haspopup="true"
                                aria-expanded="true">
                                opções <span
class="caret"></span>
                        </button>
                        <ul class="dropdown-menu"
aria-labelledby="dropdownMenu1">
                                <li><a
ng-click="deleteRuleAttribute(ruleAttribute.id)"
href="">Excluir</a></li>
                                <li><a ng-click=""
href="#/regras-atributo/{{ruleAttribute.rule.id}}">Editar</a></li>
                        </ul>
                    </div>
                </td>
        </tr>
    </table>
</div>
```

new-rule-badge.html

```
<div class="container">
```

```
      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>

      <h2>Nova regra de insígnia</h2>
      <form>

            <div class="form-group">
                  <label for="ruleName">Nome da regra</label>
                  <input type="text" ng-model="rule.name"
class="form-control" id="ruleName" placeholder="Nome">
            </div>

            <div class="form-group">
                  <label for="ruleXpReward">Recompensa em XP</label>
                  <input ng-model="rule.xp" type="number"
class="form-control" id="ruleXpReward" placeholder="Ex.: 100">
            </div>

            <div id="ruleBadgesContainer" ng-repeat="rb in
ruleBadges">
                  <div class="form-group">
                        <label for="ruleBadge"> Insígnia a ser
recompensada: </label><br>
                     <select ng-options="badge as badge.name for
badge in listBadges track by badge.id" class="form-control"
name="ruleBadge" id="ruleBadge" ng-model="ruleBadges[0].badge">
                        <option value="">Selecione uma
insígnia...</option>
                     </select>
                  </div>
            </div>

            <div class="form-group">
                  <label for="ruleName">Tipo</label>
                  <select class="form-control" ng-model="rule.type">
                     <option value="">Selecione um tipo...</option>
                     <option value="badge">Recompensa por
realização de evento</option>
                     <option value="badgeAttribute">Recompensa por
acumulo de atributo</option>
                  </select>
```

```
          </div>

          <div class="form-group" ng-show="rule.type == 'badge'">
               <label for="ruleTimeToComplete">Ocorrências
necessárias do evento para recompensa </label>
               <input ng-model="rule.timesToComplete"
type="number" class="form-control" id="ruleTimeToComplete"
placeholder="Ex.: 10">
          </div>

          <div id="badgeAttributeWrapper" ng-show="rule.type ==
'badgeAttribute'">
               <div class="form-group">
                    <label for="ruleAttribute"> Atributo:
</label><br>
                  <select ng-options="attribute as
attribute.name for attribute in listAttributes track by
attribute.id" class="form-control" name="ruleAttribute"
id="ruleAttribute" ng-model="ruleBadges[0].attribute">
                     <option value="">Selecione um
atributo...</option>
                  </select>
               </div>
               <div class="form-group">
                    <label for="ruleAttributeAmount">Quantidade
necessário para conquistar recompensa:</label>
                    <input ng-model="ruleBadges[0].goalValue"
type="number" class="form-control" id="ruleAttributeAmount"
placeholder="Ex.: 100">
               </div>
          </div>
          <div class="checkbox">
               <label>
                    <input ng-model="rule.active"
type="checkbox" ng-selected="true"> Ativar regra ao criar
               </label>
          </div>
          <button type="submit" class="btn btn-success"
ng-click="saveRule()">Salvar</button>
     </form>
</div>
```

rules-badge.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <a href="#/regras-insignia/novo" type="button" class="btn
btn-primary pull-right">Nova regra de insígnia</a>
     <table ng-table="ruleBadgeTableParams" class="table"
show-filter="false">
         <tr ng-repeat="ruleBadge in $data">

            <td title="'ID'" filter="" sortable="'id'">
                {{ruleBadge.rule.id}}
             </td>

             <td title="'Regra'" filter="{ name: 'text'}"
sortable="'name'">
                 {{ruleBadge.rule.name}}
             </td>

             <td title="'Vezes para completar'" filter=""
sortable="'timesToComplete'">
                 {{ruleBadge.rule.timesToComplete}}
             </td>

             <td title="'Recompensa em XP'" filter=""
sortable="'xp'">
                 {{ruleBadge.rule.xp}}
             </td>

             <td title="'Regra cíclila'" filter="" sortable="'repeatable'">
                 {{ruleBadge.rule.repeatable}}
             </td>

             <td title="'Finalizada'" filter="" sortable="'finished'">
                 {{ruleBadge.rule.finished}}
             </td>

             <td title="'Ativa'" filter="" sortable="'active'">
                 {{ruleBadge.rule.active}}
```

```
            </td>

            <td title="'ID da insígnia'" filter="" sortable="''">
                {{ruleBadge.badge.id}}
            </td>

            <td title="'Insígnia'" filter="" sortable="''">
                {{ruleBadge.badge.name}}
            </td>

             <td title="'Atributo'" filter="" sortable="''">
                {{ruleBadge.attribute.name || "--"}}
            </td>

             <td title="'Qtd necessária'" filter=""
sortable="''">
                {{ruleBadge.goalValue || "--"}}
            </td>

            <td>
                    <div class="dropdown">
                        <button class="btn btn-default
dropdown-toggle" type="button"
                                id="dropdownMenu1"
data-toggle="dropdown" aria-haspopup="true"
                                aria-expanded="true">
                                opções <span
class="caret"></span>
                        </button>
                        <ul class="dropdown-menu"
aria-labelledby="dropdownMenu1">
                                <li><a
ng-click="deleteRule(ruleBadge.rule.id)" href="">Excluir</a></li>
                                <li><a ng-click=""
href="#/regras-insignia/{{ruleBadge.id}}">Editar</a></li>
                        </ul>
                    </div>
                </td>
        </tr>
    </table>
</div>
```

new-rule-level.html

```
<div class="container">

      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
      <h2>Nova regra de nível</h2>
      <form>
            <div class="form-group">
                  <label for="ruleName">Nome:</label>
                  <input type="text" ng-model="rule.name"
class="form-control" id="ruleName" placeholder="Nome">
            </div>

            <div class="checkbox">
                  <label>
                        <input ng-model="rule.active"
type="checkbox" ng-selected="true"> Ativar
                  </label>
            </div>
            <br>
            <div class="form-group">
                  <label for="ruleLevelStartLevel">Começa no
nível:</label>
                  <input ng-model="ruleLevels[0].startLevel"
type="number" class="form-control" id="ruleLevelStartLevel"
placeholder="Ex.: 1">
            </div>

            <div class="form-group">
                  <label for="ruleLevelEndLevel">Termina no
nível:</label>
                  <input ng-model="ruleLevels[0].endLevel"
type="number" class="form-control" id="ruleLevelEndLevel"
placeholder="Ex.: 10">
            </div>

            <div class="form-group">
```

```
                <label for="ruleLevelXpToLevelUp">XP necessário
para avançar de nível:</label>
                <input ng-model="ruleLevels[0].xpToLevelUp"
type="number" class="form-control" id="ruleLevelXpToLevelUp"
placeholder="Ex.: 150">
           </div>

           <div id="levelRewardsContainer" ng-repeat="ra in
levelRewards">
                <div class="form-group">
                     <label for="rewardAttribute"> Atributo:
</label><br>
                    <select ng-options="attribute as
attribute.name for attribute in attributes track by attribute.id"
class="form-control" name="rewardAttribute" id="rewardAttribute"
ng-model="levelRewards[$index].attribute">
                      <option value="">Selecione um
atributo...</option>
                    </select>
                </div>
                <div class="form-group">
                     <label for="ruleAttributeAmount">Valor da
recompensa:</label>
                     <input
ng-model="levelRewards[$index].amount" type="number"
class="form-control" id="ruleAttributeAmount" placeholder="Ex.:
100">
                </div>
           </div>

           <button type="submit" class="btn btn-primary"
ng-click="addLevelReward()">Adicionar recompensa</button>
           <button type="submit" class="btn btn-success"
ng-click="saveRule()">Salvar</button>

      </form>
</div>
```

rules-level.html

```
<div class="container">
```

```
      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
      <a href="#/regras-nivel/novo" type="button" class="btn
btn-primary pull-right">Nova regra de nível</a>
      <table ng-table="ruleLevelTableParams" class="table"
show-filter="false">
          <tr ng-repeat="ruleLevel in $data">

             <td title="'ID'" filter="" sortable="'id'">
                 {{ruleLevel.rule.id}}
              </td>

              <td title="'Regra'" filter="{ name: 'text'}"
sortable="'name'">
                 {{ruleLevel.rule.name}}
              </td>

              <td title="'Ativa'" filter="" sortable="'active'">
                 {{ruleLevel.rule.active}}
              </td>

              <td title="'Inicia no nível'" filter=""
sortable="'startLevel'">
                 {{ruleLevel.startLevel}}
              </td>

              <td title="'Termina no nível'" filter=""
sortable="'endLevel'">
                 {{ruleLevel.endLevel}}
              </td>

              <td title="'XP para evoluir'" filter=""
sortable="'xpToLevelUp'">
                 {{ruleLevel.xpToLevelUp}}
              </td>

              <td>
                      <div class="dropdown">
                           <button class="btn btn-default
dropdown-toggle" type="button"
                                   id="dropdownMenu1"
```

```
data-toggle="dropdown" aria-haspopup="true"
                                aria-expanded="true">
                                opções <span
class="caret"></span>
                            </button>
                            <ul class="dropdown-menu"
aria-labelledby="dropdownMenu1">
                                <li><a
ng-click="deleteRule(ruleLevel.rule.id)" href="">Excluir</a></li>
                                <li><a ng-click=""
href="#/regras-nivel/{{ruleLevel.id}}">Editar</a></li>
                            </ul>
                        </div>
                </td>
            </tr>
        </table>
</div>
```

new-rule-xp.html

```
<div class="container">

      <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
      <h2>Nova regra de experiência</h2>
      <form>
            <div class="form-group">
                  <label for="ruleName">Nome:</label>
                  <input type="text" ng-model="rule.name"
class="form-control" id="ruleName" placeholder="Nome">
            </div>


            <div class="form-group">
                  <label for="ruleXpReward">Recompensa em
XP:</label>
                  <input ng-model="rule.xp" type="number"
class="form-control" id="ruleXpReward" placeholder="Ex.: 100">
            </div>
```

```
          <div class="form-group">
               <label for="ruleTimeToComplete">Vezes para
completar:</label>
               <input ng-model="rule.timesToComplete"
type="number" class="form-control" id="ruleTimeToComplete"
placeholder="Ex.: 10">
          </div>

          <div class="checkbox">
               <label>
                    <input ng-model="rule.repeatable"
type="checkbox" ng-selected="true"> Regra cíclica?
               </label>
          </div>

          <div class="checkbox">
               <label>
                    <input ng-model="rule.active"
type="checkbox" ng-selected="true"> Ativar?
               </label>
          </div>

          <br>

          <button type="submit" class="btn btn-success"
ng-click="saveRule()">Salvar</button>

     </form>
</div>
```

rules-xp.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <a href="#/regras-xp/novo" type="button" class="btn
btn-primary pull-right">Nova regra de experiência</a>
     <table ng-table="rulesTableParams" class="table"
show-filter="false">
```

```
          <tr ng-repeat="rule in $data">

             <td title="'ID'" filter="" sortable="''">
                 {{rule.id}}
              </td>

              <td title="'Regra'" filter="{ name: 'text'}"
sortable="''">
                 {{rule.name}}
              </td>

              <td title="'ativo'" filter="" sortable="''">
                 {{rule.active}}
              </td>

              <td title="'ocorrências necessárias'" filter=""
sortable="''">
                 {{rule.timesToComplete}}
              </td>

              <td title="'Cíclica'" filter="" sortable="''">
                 {{rule.repeatable}}
              </td>

              <td title="'recompensa em experiência'" filter=""
sortable="''">
                 {{rule.xp}}
              </td>


              <td>
                      <div class="dropdown">
                           <button class="btn btn-default
dropdown-toggle" type="button"
                                   id="dropdownMenu1"
data-toggle="dropdown" aria-haspopup="true"
                                   aria-expanded="true">
                                   opções <span
class="caret"></span>
                           </button>
                           <ul class="dropdown-menu"
```

```
aria-labelledby="dropdownMenu1">
                                    <li><a
ng-click="deleteRule(rule.id)" href="">Excluir</a></li>
                                    <li><a ng-click=""
href="#/regras-xp/{{rule.id}}">Editar</a></li>
                               </ul>
                          </div>
                     </td>
               </tr>
          </table>
</div>
```

user.html

```
<div class="container">
     <div class="alert alert-success"
ng-show="showAlert">{{alertMessage}}</div>
     <h2>Novo usuário</h2>
     <form>
          <div class="form-group">
               <label for="userName">Nome</label>
               <input type="text" ng-model="user.name"
class="form-control" id="userName" placeholder="Nome">
          </div>

          <div class="form-group">
               <label for="userExternId">ID externo</label>
               <input ng-model="user.extern_id" type="number"
class="form-control" id="userExternId" placeholder="Ex.: 10">
          </div>

          <div class="form-group">
               <label for="userLevel">Nível inicial</label>
               <input ng-model="user.level" type="number"
class="form-control" id="userLevel" placeholder="Ex.: 10">
          </div>

          <div class="checkbox">
               <label>
                    <input ng-model="user.active"
```

```
type="checkbox" ng-selected="true"> Ativo
                </label>
            </div>

            <button type="submit" class="btn btn-success"
ng-click="saveUser()">Salvar</button>

      </form>
</div>
```

index.html

```
<!DOCTYPE html>
    <!--[if lt IE 7]>      <html lang="en" ng-app="app"
class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
    <!--[if IE 7]>         <html lang="en" ng-app="app"
class="no-js lt-ie9 lt-ie8"> <![endif]-->
    <!--[if IE 8]>         <html lang="en" ng-app="app"
class="no-js lt-ie9"> <![endif]-->
    <!--[if gt IE 8]><!--> <html lang="en" ng-app="app"
class="no-js"> <!--<![endif]-->
    <head>
        <meta charset="utf-8"></meta>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge"></meta>
        <title>Spring boot and Angularjs Tutorial</title>
        <meta name="description" content=""></meta>
        <meta name="viewport" content="width=device-width,
initial-scale=1"></meta>
        <link rel="stylesheet" href="/css/style.css">
         <link rel="stylesheet"
href="/webjars/ng-table/4.0.0/ng-table.min.css">
        <link rel="stylesheet"
href="/webjars/bootstrap/3.3.6/css/bootstrap.css">
    </head>
    <body>

        <div class="home-section">
            <nav class="navbar navbar-default">
```

```
                    <div class="container-fluid">
                      <!-- Collect the nav links, forms, and other
content for toggling -->
                       <div class="navbar-header">
                         <a class="navbar-brand" href="#">Admin</a>
                       </div>

                      <div class="collapse navbar-collapse"
id="bs-example-navbar-collapse-1">
                        <ul class="nav navbar-nav">
                          <li><a href="#/usuarios">Usuários <span
class="sr-only">(current)</span></a></li>
                          <li><a href="#/atributos">Atributos<span
class="sr-only">(current)</span></a></li>
                          <li><a href="#/insignias">Insígnias<span
class="sr-only">(current)</span></a></li>
                          <li><a href="#/regras-atributo">Regras de
atributos<span class="sr-only">(current)</span></a></li>
                          <li><a href="#/regras-insignia">Regras de
insígnias<span class="sr-only">(current)</span></a></li>
                          <li><a href="#/regras-nivel">Regras de
nível<span class="sr-only">(current)</span></a></li>
                          <li><a href="#/regras-xp">Regras de
experiência<span class="sr-only">(current)</span></a></li>

                        </ul>
                      </div><!-- /.navbar-collapse -->
                    </div><!-- /.container-fluid -->
                  </nav>
        </div>

        <div ng-view></div>

        <script
src="/webjars/jquery/2.2.4/jquery.min.js"></script>
        <script
src="/webjars/bootstrap/3.3.6/js/bootstrap.min.js"></script>

        <script
src="/webjars/angularjs/1.4.9/angular.js"></script>
        <script
```

```
src="/webjars/angularjs/1.4.9/angular-resource.js"></script>
        <script
src="/webjars/angularjs/1.4.9/angular-route.js"></script>
        <script
src="/webjars/ng-table/4.0.0/ng-table.min.js"></script>


        <script src="/js/app.js"></script>
        <script src="/js/controller.js"></script>
        <script src="/js/user-controller.js"></script>
        <script src="/js/badge-controller.js"></script>
        <script src="/js/attribute-controller.js"></script>
        <script src="/js/rule-attribute-controller.js"></script>
        <script src="/js/rule-badge-controller.js"></script>
        <script src="/js/rule-level-controller.js"></script>
        <script src="/js/rule-xp-controller.js"></script>

    </body>
</html>
```