CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U. nº 198, de 14/10/2016
AELBRA EDUCAÇÃO SUPERIOR - GRADUAÇÃO E PÓS-GRADUAÇÃO S.A.

José Henrique Coelho Brandão

DESENVOLVIMENTO DE UMA PLATAFORMA GAMIFICADA PARA O LOGIC LIVE

Palmas – TO

2019

José Henrique Coelho Brandão

DESENVOLVIMENTO DE UMA PLATAFORMA GAMIFICADA PARA O LOGIC LIVE

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso I (TCC I) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientadora: Profª. D.ra Parcilene Fernandes de Brito.

Palmas – TO

2019

José Henrique Coelho Brandão

DESENVOLVIMENTO DE UMA PLATAFORMA GAMIFICADA PARA O LOGIC LIVE

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso I (TCC I) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientadora: Profª. D.ra Parcilene Fernandes de Brito.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. D.ra Parcilene Fernandes de Brito

Orientadora

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Fabiano Fagundes

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Jackson Gomes de Souza

Centro Universitário Luterano de Palmas – CEULP

Palmas – TO

2019

RESUMO

BRANDÃO, José Henrique Coelho. **Desenvolvimento de uma Plataforma Gamificada para o Logic Live.** 2019. 74 f. Trabalho de Conclusão de Curso (Graduação) - Curso de Bacharelado em Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas/TO, 2019.

Este trabalho tem por objetivo empregar uma arquitetura baseada em serviços no desenvolvimento de uma plataforma gamificada, a fim de permitir a inclusão e reutilização de componentes, além de disponibilizar a análise de desempenho de cada jogador nos mais diversos módulos do Logic Live. No referencial teórico são apresentados uma visão geral das técnicas de arquitetura de software, dando ênfase para arquitetura orientada a serviços e sua forma de comunicação através da Internet que possibilita disponibilizar funcionalidades, dados e recursos em ambiente de computação distribuída na forma de serviços. Também são apresentados as definições e elementos que compõem a gamificação, além de discorrer sobre o conceito de Plataformas de Gamificação. A metodologia do trabalho é formada por um conjunto de etapas que correspondem ao desenvolvimento da Arquitetura, Documentação, Implementação e Inserção do Módulo Tabela Verdade na Plataforma Gamificada. O primeiro resultado do trabalho refere-se ao desenvolvimento da Arquitetura com componentes desacoplados e organizado em domínios altamente reutilizáveis. O segundo resultado apresenta a Documentação da API que auxilia os desenvolvedores a entender e consumir os recursos da plataforma. O terceiro resultado demonstra a Implementação da plataforma e o consumo da API para instituir as páginas dos jogadores e gestão. O quarto resultado apresenta o processo de Inserção do Módulo Tabela Verdade utilizando a documentação da Plataforma Gamificada.

Palavras-chave: Arquitetura de *Software,* API REST, Plataforma Gamificada.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - APPLICATION PROGRAMMING INTERFACE

IBGE - INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA

FTP - FILE TRANSFER PROTOCOL

HTML - HYPERTEXT MARKUP LANGUAGE

JSON - JAVASCRIPT OBJECT NOTATION

MVC - MODEL-VIEW-CONTROLLER

REST - REPRESENTATIONAL STATE TRANSFER

SOAP - SIMPLE OBJECT ACCESS PROTOCOL

SMTP - SIMPLE MAIL TRANSFER PROTOCOL

UDDI - UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION

URI - UNIFORM RESOURCE IDENTIFIER

URL - UNIFORM RESOURCE LOCATOR

WEB - WORLD WIDE WEB

WSDL - WEB SERVICES DESCRIPTION LANGUAGE

W3C - WORLD WIDE WEB CONSORTIUM

XML - EXTENSIBLE MARKUP LANGUAGE

SUMÁRIO

1. INTRODUÇÃO

Com o surgimento de novas Tecnologias da Informação e Comunicação (TICs), foi possível estender a sua utilização a diversos campos de atuação, como, para o campo da Educação, que ao longo do tempo tem agregado novas metodologias pedagógicas baseadas em TICs, com o intuito de melhorar a experiência de aprendizagem do aluno. O uso de tecnologias interativas, como jogos digitais e seus processos de gamificação, possui a competência de colaborar e aperfeiçoar as metodologias existentes (PAULA; VALENTE; BURN, 2014).

O termo gamificação (do original em inglês *gamification*), formulado por Nick Pelling em 2002, refere-se ao uso de elementos de jogos (como mecânicas, dinâmicas e estéticas), com a finalidade de estimular o interesse e melhorar a experiência do usuário nas resoluções de problemas e ensino (VIANNA et al., 2013). No início da segunda década do século XXI, o termo ganhou mais popularidade no mercado ao redor do mundo, sendo utilizado nas mais diversas áreas, como educação, saúde, políticas públicas e treinamento de profissionais para ganho de produtividade (VIANNA et al., 2013).

Com a popularização dos processos de gamificação, surgiu a necessidade de desenvolver plataformas gamificadas que possam resolver diferentes tipos de problemas como, por exemplo, gerar informações em tempo real para avaliar o desempenho dos usuários, e estar em constante evolução para operar em ambientes distintos. Atualmente empresas como Gamify, Bunchball, Badgeville, Ludos Pro, entre outras, fazem uso de plataformas para, por meio de *websites*, *Widges* e APIs, fornecerem aplicações gamificadas para milhares de pessoas ao redor do mundo (STEFANONI; STAGLIANÒ, 2014).

Além de utilizar mecânicas de jogos, como conquistas, missões e prêmios, com a finalidade de estimular o jogador a realizar uma tarefa, uma plataforma gamificada possibilita, por meio de métricas, relatórios e *feedback*, a realização de análises de desempenho, engajamento e crescimento de cada jogador ao longo do jogo (PRO, 2019).

Se antes os jogos eram taxados de inimigos dos estudos e da produtividade, hoje tem-se na gamificação uma relevante metodologia para estimular pessoas a ter um melhor rendimento, tanto escolar, quanto profissional. Por meio das plataformas gamificadas é possível uma análise detalhada das conquistas e dificuldades de cada pessoa, como também estabelecer um processo de ensino baseado na forma de aprendizagem de cada um. Com estas análises, é possível ainda avaliar se a gamificação está de fato auxiliando na resolução do problema para qual a plataforma foi proposta. Segundo Vianna et al (2013), monitorar a

motivação dos jogadores e mensurar as métricas, são iniciativas indispensáveis para o sucesso da gamificação.

No entanto, o processo de desenvolvimento dos recursos que possibilitem monitorar o engajamento dos jogadores ao longo do jogo, pode ser demorado e oneroso (ZICHERMANN; CUNNINGHAM, 2011). Logo a utilização de tecnologias, como *Web Services* que viabiliza a reutilização dos recursos de análise da mecânica do jogo, passa a ser fundamental para a implementação plataforma gamificada.

Na Arquitetura Baseada em Serviço (SOA), os serviços possuem suas funcionalidades na plataforma assumindo o papel de recursos independentes e com baixo acoplamento. Podendo ser disponibilizado por demanda para um usuário final ou outro serviço, sem a necessidade de interação humana e geralmente implementados em padrões de comunicação que utilizam a Internet como meio de conexão (AL-KOFAHI, 2011; STAL, 2002).

Nesse contexto, Degan (2015) diz que uma arquitetura baseada em serviços permite o compartilhamento de lógica de negócios e métodos entre aplicações distintas. Utilizar este tipo de arquitetura facilita a reutilização de componentes e análise de dados nos diversos módulos da plataforma (SRIPADA; REDDY; KHANDELWAL, 2016).

Nesse sentido, por meio de *Web Services* é possível disponibilizar funcionalidades em ambiente de computação distribuída na forma de serviços, Isso que permite oferecer para outras aplicações e serviços, dados e recursos de forma eficiente e rápida, além de viabilizar, por meio de padrões de comunicação, a interoperabilidade entre aplicações desenvolvidas em linguagens de programação diversas ou por diferentes fornecedores (HANSEN ET. AL. 2003, KREGER 2001).

A maioria dos serviços disponíveis na Internet utiliza o padrão de comunicação SOAP (MATOS, 2009). Segundo W3c (2019), o SOAP oferece um modelo genérico capaz de trocar informações entre aplicações descentralizadas. Contudo o seu estilo foi projetado com excesso de regras e para utilizar os recursos da Web de forma mais efetiva o conceito de *Representational State Transfer* (REST) foi desenvolvido, possibilitando a troca de informações de forma sucinta e com menos padrões.

Por seu estilo simples de comunicação, o conceito REST vem sendo utilizado por grandes empresas de tecnologia ao redor do mundo (SILVA; GONÇALVES, 2012). Fielding (2000) defende a utilização do padrão REST que permite facilitar a interação com outras

tecnologias, utilizando o protocolo HTTP baseado na Web. Assim, com o REST é possível maximizar a troca de informações, além de disponibilizar uma alta escalabilidade do sistema. Por isso, em geral, o estilo REST é aplicado no desenvolvimento de sistemas distribuídos baseado na Web (FIELDING, 2000).

Conforme Sripada, Reddy e Khandelwal (2016), implementar uma plataforma gamificada utilizando arquitetura de serviços com estilo REST facilita a separação dos componentes da plataforma possibilitando o desenvolvimento de módulos altamente desacoplados, tornando fácil a inclusão de novos componentes, conforme necessário (SRIPADA; REDDY; KHANDELWAL, 2016).

O projeto Logic Live, desenvolvido pelo Grupo de Estudos em Novas Tecnologias para processos de Ensino e Aprendizagem (GENTE), é composto por uma série de módulos que visa o ensino da Lógica.

Neste trabalho, foi empregada a arquitetura de serviços, seguindo o padrão mais adequado para desenvolver uma plataforma gamificada que viabilizou a análise dos jogadores, integração e reutilização de componentes dos módulos, inclusão de novos módulos e elementos voltados ao processo de ensino e aprendizagem da disciplina de Lógica (Logic Live).

Para atingir a proposta do trabalho, foi elaborado um conjunto de etapas necessárias para o desenvolvimento da plataforma gamificada, logo em seguida uma arquitetura baseada em serviços SOA foi implementada e a plataforma de integração foi desenvolvida, disponibilizando uma documentação para que os módulos possam ser integrados. Por fim, a inclusão do Módulo de Tabela Verdade à plataforma gamificada foi realizado. O módulo é composto por mecanismo e elementos de jogos que auxiliam na aprendizagem de conteúdos da disciplina de lógica, especialmente na validação das fórmulas da Lógica Proposicional seguindo o método da Tabela Verdade.

Por fim, o presente trabalho é desenvolvido da seguinte forma: seção 2, Referencial Teórico, são apresentados os conceitos da Arquitetura de *Software*, a forma de comunicação na Web baseada em serviços, os padrão de arquitetura de serviços na Web e uma breve demonstração sobre os conceitos de gamificação e plataformas de gamificação; seção 3, Materiais e Métodos, fornece uma explanação referente aos materiais e tecnologias utilizadas e apresenta a metodologia realizada para o desenvolvimento do trabalho; seção 4, Resultados, apresenta de forma detalhada os processos e resultados obtidos na execução do trabalho;

seção 5, Considerações Finais, apresenta as conclusões referentes ao desenvolvimento deste trabalho e melhorias que podem ser feitas em trabalhos futuros, em seguida são listadas as referências que serviram como base para a elaboração do trabalho.

2. REFERENCIAL TEÓRICO

Esta seção apresenta uma visão geral das técnicas utilizadas na Arquitetura de *Software* (subseção 2.1), demonstra a forma como a arquitetura é aplicada em serviços (subseção 2.2), difunde os conceitos de serviços baseado na Web (subseção 2.2.1), apresenta o conceito e padrões de arquitetura de serviços na Web (subseção 2.2.2), discorre uma visão geral das definições da Gamificação (subseção 2.3), apresenta os elementos que compõem a gamificação (subseção 2.3.1), e expõem o conceito de Plataformas de Gamificação (subseção 2.4).

2.1 ARQUITETURA DE SOFTWARE

Com a evolução das tecnologias de desenvolvimento de software, surgiu a necessidade de criar novos meios capazes de possibilitar a utilização de todo o potencial das atuais técnicas de forma organizada e eficiente. As pesquisas no campo da Arquitetura de *Software* permitiram evoluções neste sentido. Fielding (2000) definiu em sua tese de doutorado uma terminologia para Arquitetura de *Software* baseada em um conjunto de elementos. Uma síntese dessa terminologia é apresentada a seguir (FIELDING, 2000):

- Abstração em tempo real: uma arquitetura de *software* envolve a abstração, estruturação e relacionamento entre os componentes em tempo de execução durante alguma fase de sua operação. Agrega-se a isso, que pode haver níveis de abstração, compostos por diversos elementos do sistema, cada um com características, limitações e arquitetura própria. Para Allen e Garlan (1997), é necessário manter um monitoramento contínuo no modelo de tempo de execução, levando em consideração a finalidade e o custo-benefício para que este possa, de fato, refletir as alterações relevantes do sistema e, para que isso seja possível, é preciso saber quais informações são necessárias obter e representar no modelo.
- Elementos: segundo Perry e Wolf (1992), uma arquitetura de *software* é composta por um conjunto de elementos que fornecem os recursos necessários para criar a lógica da arquitetura. Os elementos abrangem os dados, processamento e conexão com os demais componentes. As conexões entre os elementos são definidas pelas suas propriedades e restrições. Os elementos podem ser classificados em: Componentes, “unidade abstrata de instruções de *software* e estado interno que fornece uma transformação de dados através de sua interface” (FIELDING, 2000, p. 9), p.ex.: a

abstração das funcionalidades de um componente de cadastro que captura as informações do usuário; Conectores, “mecanismo abstrato que media a comunicação, coordenação ou cooperação entre os componentes” (FIELDING, 2000, p. 10), p. ex.: a forma como a interação do componente de cadastro e de autenticação é constituída**;** Dado, “elemento de informação que é transferido de um componente, ou recebido por um componente, através de um conector” (FIELDING, 2000, p. 11), p. ex.: as informações do usuário enviadas do componente de cadastro para o componente de autenticação.

- Configurações: organização das conexões entre dados, conectores e componentes no ciclo de execução do sistema. Especificamente, uma configuração é um conjunto de regras próprias nas conexões entre os componentes. Por exemplo, um grafo de componentes e conectores que descrevem a estrutura de relações da arquitetura.
- Propriedades: agregação de todas as propriedades do sistema que resultam da seleção e organização de componentes, dados e conectores. Conforme defende Bass et. al. (1998), as propriedades agregadas pelo sistema e suas configurações propiciam a evolução, reutilização de componentes, eficiência e qualidade durante o tempo de execução do sistema. Segundo Ghezzi et. al. (1991), as propriedades são definidas por meio dos conjuntos de relações inerentes a uma arquitetura, seguindo os princípios de engenharia de *software*. Por exemplo, um conjunto de validação de tipo de dado, alcança a qualidade de ser reutilizável em diversos componentes por sua forma genérica. Assim, a forma uniforme acaba por criar interface de componentes de um único tipo. Logo, conforme Fielding (2000, p. 13), “as relações de uma arquitetura é a forma uniforme das interfaces de componentes, que são determinadas pelo princípio da semelhança, com a finalidade de alcançar qualidades que se tornarão propriedades da arquitetura, como componentes reutilizáveis e configuráveis”. Portanto, uma arquitetura tem como finalidade criar um conjunto de propriedades que agrupadas implementam a estrutura e organização da lógica dos requisitos do sistema.
- Estilos: de acordo com Nitto e Rosenblum (1999), estilos são uma forma de agrupar as características comuns de cada arquitetura, pois com diferentes propriedades, pode ser difícil relacionar sistemas em ambientes distintos. Para Perry e Wolf (1992) e Shaw (1995), um estilo é uma abstração de diferentes tipos de elementos e aspectos da arquitetura, não sendo necessária a definição de um estilo genérico para todas as

arquiteturas. Um estilo pode reunir regras e relacionamentos entre seus elementos e será mais adequado na medida em que atender às necessidades específicas dos requisitos do sistema.

- Padrões de linguagem: além de estilos, a engenharia de *software*, principalmente na programação orientada a objetos, tem aprofundado o uso de padrões de linguagem para desenvolver concepções relativas ao sistema baseado em objetos. Conforme Gamma et. al. (1995), os padrões são responsáveis por definir uma orientação para o uso de técnicas de programação orientada a objetos, como classes, herança e interfaces. Entretanto, Monroe et. al. (1997) argumenta que o principal benefício dos padrões é elaborar protocolos parcialmente complexos de relacionamentos entre os objetos de uma única abstração. Um padrão pode ser considerado como um conjunto de métodos que implementados gera um processo que tem como objetivo resolver um problema definido pela arquitetura.
- Visões: Como no sistema pode ter várias arquiteturas e estilos, é necessário visualizar a arquitetura de muitas formas diferentes. Para Perry e Wolf (1992), uma arquitetura de *software* pode ter três formas importantes de visualização: processamento, dados e conexão. Na visualização do processo é ressaltado o fluxo dos dados através dos componentes e alguns pontos das relações entre componentes e aos dados. Já a visualização de dados foca no fluxo do processamento, com pouco destaque para os conectores. Por fim, a visualização de conexão evidencia o relacionamento entre os componentes e sua comunicação.

2.2 ARQUITETURA ORIENTADA A SERVIÇOS

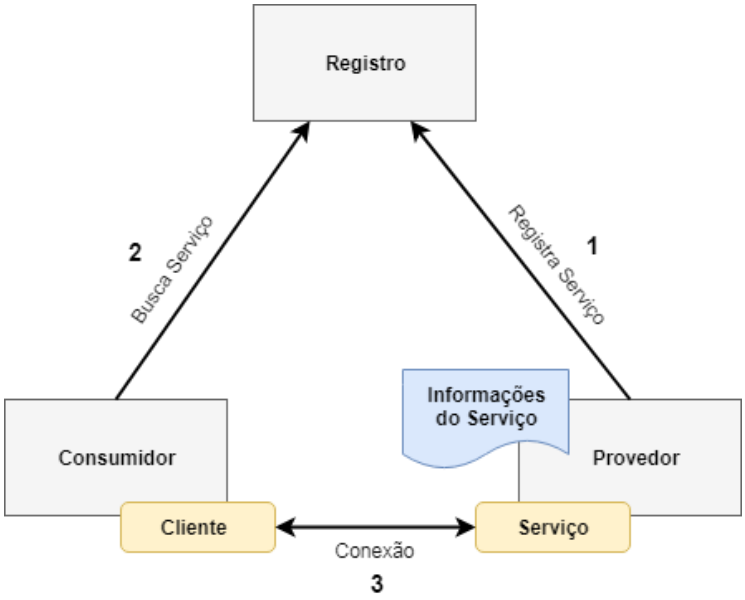
Na Arquitetura de *Software* são definidos os componentes, suas propriedades e relacionamentos com o *software*. Entretanto, com o avanço da tecnologia e o surgimento de novas linguagens de programação, mostrou-se necessário desenvolver uma nova forma de arquitetura que se utiliza da rede de computadores para distribuir recursos entre tecnologias distintas.

Arquitetura Orientada a Serviços (SOA) é um conjunto de serviços ou recursos independentes e com baixo acoplamento. Cada conjunto dispõe de uma interface própria que auxilia o serviço nas relações que estabelece com outros serviços fora do seu escopo e com implementações ou tecnologias diferentes. A disponibilidade do recurso é feita sob demanda

para um usuário final, aplicativo ou para outro serviço, sem a necessidade de interação humana. Cada serviço possui sua funcionalidade, podendo ser simples, que acopla um único serviço; ou complexa, quando um serviço recebe a solicitação e distribui para vários outros serviços. A rede SOA é geralmente implementada em tecnologias que utilizam a Internet com meio de conexão, mas não significa que diferentes tipos de tecnologias de conexão possam ser empregados (AL-KOFAHI, 2011; STAL, 2002).

Para que esses serviços ou recursos possam comunicar-se, é necessário um padrão de comunicação. Logo a principal característica de uma arquitetura SOA é utilizar três tipos de agentes de software como padrão para as comunicações: os provedores de serviço, os consumidores de serviço e o registro de serviço (HUHNS; SINGH, 2005). As relações e procedimentos entre os agentes é apresentada na Figura 1.

Figura 1 - Arquitetura conceitual orientada a serviço.



Fonte: Figura adaptada de Systinet (2002)

Conforme apresentado na Figura 1, a SOA pode ser dividida em três operações básicas: provedor, consumidor e registro. O provedor é responsável por tornar disponível e fornecer a descrição do serviço. O consumidor realiza a consulta ao registro que encontra o serviço compatível. O registro envia ao consumidor instruções sobre o caminho e descrição do serviço. Por fim, o consumidor encontra o serviço através do registro e vincula o cliente ao serviço (SYSTINET, 2002).

Dessa forma, uma arquitetura SOA opera como, por exemplo, um classificado de carros *on-line*. O vendedor (provedor) anuncia um carro (serviço) informando suas características (marca, ano, cor, valor etc.) no classificado. As informações são estruturadas

pelo sistema do classificado (registro). Um comprador (consumidor) procura por um carro (serviço) fornecendo as características ao site. O sistema faz a busca do veículo (serviço) e informa que existe um carro (serviço) com as características fornecidas e envia as informações de contato para o comprador (consumidor) iniciar a comunicação sobre a compra do carro (serviço) com o vendedor (provedor).

2.2.1 WEB SERVICES

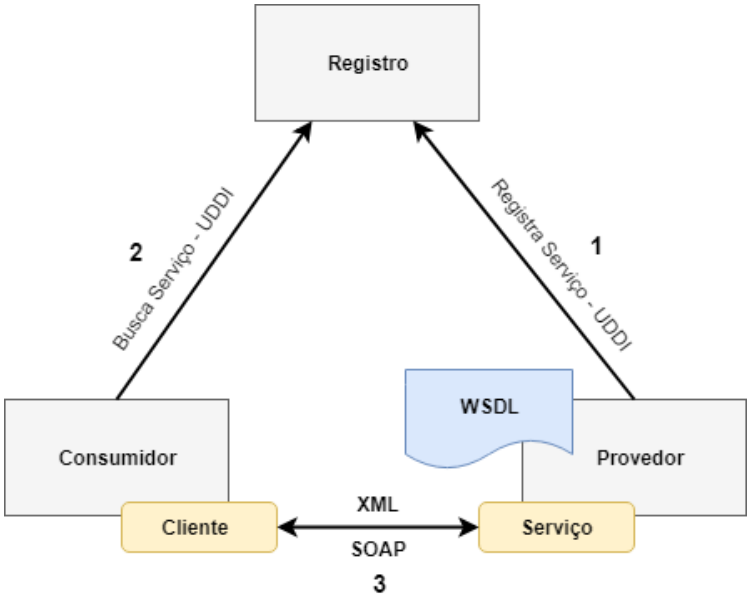
Web Services são uma coleção de aplicações distintas que tem como base a Arquitetura Orientada a Serviços para disponibilizar e executar os seus recursos sobre uma rede, em geral a Internet, ou seja, podem ser compreendidos como uma interface composta por operações disponíveis na Web através de padrões de mensagens, geralmente em formato XML (HANSEN ET. AL. 2003, KREGER 2001). Estas coleções podem ser desde simples aplicações com poucos processos, até complexos conjuntos de sistemas. Nesse sentido, por meio de *Web Services* é possível disponibilizar para outras aplicações e serviços, dados e recursos de forma eficiente e rápida (HANSEN ET. AL. 2003, KREGER 2001).

O principal objetivo do *Web Service* é disponibilizar funcionalidades em ambiente de computação distribuída na forma de serviços. Com isso, é possível permitir interoperabilidade entre aplicações desenvolvidas em linguagens de programação diversas ou por diferentes fornecedores.

Um aspecto básico dos *Web Services* é a possibilidade de utilizar diversos formatos de transmissão de dados na rede de computadores. Portanto, é possível transmitir os dados por meio de vários protocolos, tais como HTTP, SMTP, FTP ou protocolos proprietários (W3C, 2019).

Os três principais componentes funcionais da grande maioria de serviços disponíveis na Internet são: transporte, descrição e descoberta (SYSTINET, 2002). Na arquitetura de *Web Service*, os três componentes são implementados utilizando SOAP, WSDL e UDDI, respectivamente. O conceito de arquitetura SOA empregada na tecnologia *Web Service* é apresentado na Figura 2.

Figura 2 - Arquitetura conceitual SOA com SOAP, WSDL e UDDI.



Fonte: Figura adaptada de Systinet (2002)

Na arquitetura SOA para *Web Service*, o UDDI adota a funcionalidade de registro. As instruções de registro e consulta são implementadas utilizando o protocolo da UDDI. O documento WSDL descreve as funcionalidades do serviço para ser usado no relacionamento entre o cliente e o serviço (MATOS, 2009). As funções de transporte são implementadas sobre o protocolo SOAP.

O protocolo SOAP utiliza tecnologia XML para estruturar mensagem, oferecendo um modelo genérico capaz de trocar informações entre ambientes distribuídos e descentralizados com tecnologia e implementações distintas (W3C, 2019). Conforme Streibel (2005) esse protocolo fornece uma arquitetura para comunicação de forma transparente entre sistemas utilizando documentos XML. A Figura 3 e a Figura 4 apresentam um exemplo de documento SOAP de solicitação e resposta respectivamente.

Figura 3 - Estrutura de um documento SOAP com invocação ao método GetTemperatura

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.tempoagora.com.br/cidade">
  <m:GetTemperatura>
    <m:Cidade>Palmas, TO</m:Cidade>
  </m:GetTemperatura>
</soap:Body>

</soap:Envelope>
```

A Figura 3 apresenta um documento SOAP de solicitação ao um serviço de previsão de tempo. O documento é estruturado e codificado no padrão XML, sua construção segue a estrutura:

- *Envelope*: elemento raiz que identifica o arquivo XML como sendo uma mensagem SOAP e define os *namespaces* utilizados pela mensagem;
- *Body*: armazena as informações sobre o protocolo, localização, métodos e parâmetros das consultas ou respostas do serviço;
- *GetTemperatura*: método responsável por realizar a invocação do serviço;
- *Cidade*: campo que contém o nome da cidade para ser enviado na consulta.

Figura 4 - Estrutura do documento de resposta SOAP

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.tempoagora.com.br/cidade">
  <m:GetTemperaturaResposta>
    <m:Temperatura>37</m:Temperatura>
  </m:GetTemperaturaResposta>
</soap:Body>

</soap:Envelope>
```

Conforme apresentado na Figura 4, a estrutura de resposta do documento SOAP segue o padrão de codificação da solicitação.

2.2.2 REST

Apesar do protocolo SOAP ser uma tecnologia acessível por diversas linguagens de programação, alguns aspectos da tecnologia foram projetados em excesso (SILVA; GONÇALVES, 2012). Para resolver o problema dos excessos surgiu o REST, um novo estilo mais sucinto, com menos padrões e que pode utilizar os recursos da Web de forma mais efetiva.

Com sua forma mais sucinta de desenvolver, o conceito REST vem ganhando espaço entre a comunidade de desenvolvedores e atualmente empresas como Google, Facebook, Twitter e Amazon empregam a tecnologia Web *Service* de estilo REST para consumo dos seus serviços (SILVA; GONÇALVES, 2012).

O conceito REST foi apresentado pela primeira vez por Roy Fielding (2000) em sua tese de doutorado. O conceito define um conjunto de vários estilos arquiteturais que restringe as funcionalidades e o grau de relação entre os componentes da arquitetura (FIELDING, 2000). A definição de estilos permite construir técnicas, regras e padrões a fim de melhorar o projeto de desenvolvimento de *software*, além de facilitar a integração com outras tecnologias devido aos padrões utilizados. Assim o conjunto de estilo REST pode ser aplicado no desenvolvimento de sistemas distribuídos baseado na Web (FIELDING, 2000).

REST não é um protocolo ou tecnologia, é um conjunto de estilos arquitetural que, após implementados, trazem vantagens e restrições, limitando o papel de cada componente e as relações dentro de um *software* (ZUR MUEHLEN; NICKERSON; SWENSON, 2005).

REST foi idealizado para lidar ao máximo com a arquitetura da Web, utilizando os protocolos e tecnologias já em uso, sendo o principal o HTTP, que é utilizado diariamente por bilhões de pessoas ao redor do mundo no acesso a uma página Web, envio de arquivos, entre outros recursos.

Com REST é possível minimizar a latência e a troca de informações pela Web, além de fortalecer a escalabilidade dos componentes. O REST promove acesso aos recursos em forma de páginas utilizado *hyperlinks* que representam o estado de um *software*, assim o cliente tem a percepção de está interagindo diretamente com os estágios da aplicação ao navegar pelos *links* do REST (MATOS, 2009).

O principal conceito da Arquitetura REST está relacionado ao uso de recursos que fornece um conjunto de interfaces genéricas para as interações entre os componentes da

arquitetura. Um recurso pode ter diversos formatos de representação, como por exemplo JSON, XML e HTML.

Segundo (Fielding, 2000) um recurso é qualquer informação (documento, imagem ou serviço) que possa ser nomeada. O recurso é um mapeamento conceitual para um conjunto de entidades, não a entidade que corresponde ao mapeamento em tempo específico.

O estilo REST estabelece que todo recurso deve possuir um identificador único, que será empregado como interface para acesso ao recurso. Um *Web Service* que emprega o estilo REST utiliza URI como identificador único para mapear seus recursos, em conjunto com o protocolo HTTP e seus métodos para realizar operações de criação (POST), consulta (GET), atualização (PUT) e destruição de dados (DELETE) (RODRIGUEZ, 2008). Além dos métodos o protocolo HTTP institui uma série de códigos de resposta para informar quando ocorre sucesso, alterações de URI ou erro ao solicitar um recurso. Os principais códigos são exibidos na Tabela 1 (SILVA, 2017).

Tabela 1- Tabela dos principais códigos de resposta do protocolo HTTP

CÓDIGO	DESCRIÇÃO
100	Resposta provisória usada para informar que a solicitação foi recebida, porém seu processamento está em espera. Após concluir o processo o servidor deve enviar uma nova resposta.
200	Indica que a solicitação foi recebida, processada e as informações da resposta enviada.
201	A solicitação foi atendida e resultou na criação de um novo recurso.
300	Informa que há mais de uma URI para a solicitação. Uma lista com os URIs é retornada.
301	Esse código de resposta significa que a URI do recurso solicitado foi alterado permanentemente para uma nova URI. A nova URI deve ser informada na resposta.
302	O código indica que a URI do recurso foi alterada temporariamente. O cliente deve continuar a usar a URI de solicitação para solicitações futuras.
304	Essa resposta é usada para informar que o recurso não foi modificado. Logo o cliente pode utilizar a versão do recurso em cache.

400	Código de resposta informando que a solicitação não foi atendida devido ao um erro de sintaxe. O cliente deve corrigir o erro e repetir a solicitação.
401	Indica que a solicitação requer autenticação. O cliente deve se autenticar e repetir a solicitação com um campo de cabeçalho contendo as informações de autenticação.
403	O servidor atendeu à solicitação, mas está se recusando a retornar a resposta devido a direitos de acesso.
404	O servidor não encontrou o recurso solicitado correspondente a URI informada.
411	O servidor recusou a solicitação porque o tamanho da mensagem não foi informado.
415	Indica que a solicitação foi recusada porque o formato da resposta solicitada não é suportado pelo servidor.
429	Informa que o limite que solicitação num dado tempo foi atingido.
500	Esta resposta significa que o servidor encontrou uma condição inesperada que impediu de processar a solicitação.
501	O código de resposta é retornado quando o método da solicitação não é suportado pelo servidor.
503	O código é usado para informar que há uma sobrecarga ou manutenção temporária do servidor, portando a solicitação não pode ser atendida.
505	Esta resposta indica que o servidor não suporta a versão do HTTP usada na solicitação.

Fonte: The Internet Society (1999)

Utilizando o exemplo de um serviço de previsão do tempo que emprega o estilo REST no seu Web *Service*, é possível realizar o mapeamento de recursos da seguinte forma: considerando que o serviço tem a funcionalidade de fornecer uma lista de cidades com previsões do tempo disponíveis, e que está alocado em um servidor Web qualquer, a URI para acessar a lista seria por exemplo “http://tempoagora.com.br/api-tempo/cidades”. Sendo

que, http é o protocolo, “tempoagora.com.br” é o domínio do servidor onde o serviço está alocado, “api-tempo” é o nome do serviço, e “cidades” é o recurso acessado. Ao consumir essa URI é retornada uma lista de cidades com previsões do tempo disponíveis.

Para acessar a previsão detalhada de uma determinada cidade, levando em consideração que o serviço utiliza o código IBGE das cidades como identificador único, a URL é alterada e passa a receber o código da cidade, ficando da seguinte forma: “http://tempoagora.com.br/api-tempo/cidades/1721000”, onde “1721000” é código da cidade que deseja obter a previsão detalhada.

Seguindo o estilo de mapeamento de recursos, uma solicitação de previsão de uma cidade para um determinado horário, a URI seria semelhante a “http://tempoagora.com.br/api-tempo/cidades/1721000/19”, onde “19” é o horário que deseja realizar a previsão.

2.3 GAMIFICAÇÃO

Conforme Deterding (2011), a Gamificação pode ser entendida como a utilização de elementos de *design* de jogos em contextos de “não-jogos”. Conforme Hagglund (2012), o termo Gamificação foi citado pela primeira vez em 2002 pelo programador de jogos Nick Pelling, e desde foi definida como sendo uma estratégia apoiada na aplicação de elementos de jogos para atividades que não possuem o propósito de entretenimento.

Hagglund (2012) aponta que o termo Gamificação não recebeu a devida atenção no campo empresarial logo após sua definição, pois não despertou interesses entre as organizações. Conforme a expansão do termo anos depois, ganhou popularidade em segmentos da indústria de jogos e diversos produtos gamificados foram desenvolvidos no âmbito comercial. Até mesmo em eventos científicos passou a ser tema de estudos, de modo que um crescente espaço foi sendo ocupado e aplicações gamificadas foram aumentando (COSTA; MARCHIORI, 2015).

Nos últimos anos, os elementos de *design* de jogos têm sido aplicados em campos variados. Conforme Costa e Marchiori (2015), no âmbito empresarial, a gamificação tornou-se popular por proporcionar envolvimento e produtividade aos funcionários em suas atividades. No campo da Educação é útil para o engajando de estudantes nas atividades escolares, e também em sistemas de *e-learning* (sistemas de ensino eletrônico); já no campo

da saúde auxilia na contenção de custos e em programas de obesidade (VIANNA et al., 2013).

Segundo Azarite (2013), as atividades que apresentam características que envolvam caráter lúdico ou de desafio, e elementos do *design* de jogos, podem ser consideradas como atividades gamificadas. Por ser considerada como estratégia válida para estimular emoções positivas, ou atrelar recompensas ao cumprimento de tarefas, a gamificação mostra-se útil também no reforço de comportamentos de usuários de sistemas (MENEZES et al., 2014). Nesse sentido, para entender o desenvolvimento de sistemas gamificados, a subseção 2.1.1 apresentará de forma detalhada os elementos de jogos que podem ser considerados para a criação de ambientes gamificados.

2.3.1 Elementos de Jogos na Gamificação

Quanto aos elementos de jogos, Werbach e Hunter (2012) identificaram dinâmicas, mecânicas e componentes como categorias aplicáveis ao desenvolvimento e estudos na gamificação. Essas categorias podem ser organizadas em ordem decrescente de abstração, de forma que cada mecânica esteja vinculada a uma ou mais dinâmicas, e cada componente também seja vinculada a uma ou mais mecânicas ou dinâmicas (KUUTTI, 2013).

Sobre as dinâmicas de jogos, estas consistem nas emoções, ou no resultado do sentimento de desejos e motivações; são também os temas em torno do qual o jogo se desenvolve (BUNCHBALL, 2010). De acordo com Wood e Reiners (2015),

> As dinâmicas são os comportamentos e interações resultantes da interação dos usuários com os componentes e mecanismos. Assim, dependem da natureza e experiência dos usuários. O usuário introvertido com aversão ao risco se comporta de maneira diferente em comparação a alguém com afinidade para explorar situações de risco. Assim, o design da dinâmica do jogo deve incorporar os atributos dos usuários, que devem ser atualizados durante todo o progresso dentro do sistema gamificado (WOOD; REINERS, 2015, p. 3042).

Para Werbach e Hunter (2012), neste sentido, as dinâmicas estão no mais alto nível de abstração de elementos de jogos. A Tabela 2 apresenta uma classificação das dinâmicas a partir do modelo apresentado em Werbach e Hunter (2012) e Wood e Reiners (2015).

Tabela 2 - Dinâmicas de Jogos

DINÂMICA	DESCRIÇÃO

Narrativas	Um enredo consistente e contínuo que pode ser estendido por vários estágios ou níveis.
Progressão	Expressa o crescimento e indica a evolução no decorrer do tempo dos jogadores.
Relacionamentos	São interações que os jogadores têm uns com os outros gerando sentimentos de camaradagem, status e altruísmo.
Emoções	São as percepções dos jogadores como curiosidade, competitividade, frustração e felicidade.
Restrições	Limitações ou características impostas aos jogadores.

Fonte: Werbach e Hunter (2012); Wood e Reiners (2015)

A Tabela 2 apresenta as descrições de algumas dinâmicas de jogos. As narrativas (ou histórias) fornecem um enredo contínuo e convincente aos jogos, atribuindo contexto e significado para as interações e aventuras dos usuários. Já a progressão apresenta o crescimento e desenvolvimento do usuário com o passar do tempo, permitindo o seu acompanhamento e fornecendo *feedback* adequado (WOOD; REINERS, 2015).

Os relacionamentos incluem um conjunto de interações que levam a ligações emocionais entre os usuários, por exemplo, o envio de presentes virtuais com o intuito de fazer novos amigos. As emoções que são experimentadas pelos usuários envolvem um senso de curiosidade ou competitividade, que pode ser projetado no sistema com o intuito de alcançar os resultados desejados (WOOD; REINERS, 2015). Por fim, as restrições correspondem às regras impostas aos usuários capazes de limitar suas ações no sistema, fazendo com que encontrem caminhos alternativos para alcançar a meta definida.

As mecânicas de jogos podem ser caracterizadas como as regras que orientam as ações dos jogadores em uma direção desejada, as recompensas que compõem o jogo, como também os aspectos que o torna desafiador, divertido e satisfatório (COSTA; MARCHIORI, 2015). Podem ser identificadas dez importantes mecânicas de jogos, apresentadas na Tabela 3.

Tabela 3 - Mecânicas de Jogos

MECÂNICA	DESCRIÇÃO

Desafios	Atividades ou tarefas que exigem esforço para serem resolvidas.
Acaso	Elementos de aleatoriedade.
Competição	Os jogadores concorrem entre si, um jogador (ou grupo) ganha e o outro perde.
Cooperação	Promove a interação entre os usuários, pois os jogadores devem trabalhar juntos para alcançar um objetivo comum.
Feedback	Informações sobre o estado do jogador.
Aquisição de Recursos	Obtenção de itens úteis ou colecionáveis.
Recompensas	Vantagens por alguma ação ou realização.
Transações	Negociação entre jogadores, diretamente ou através de intermediários.
Turnos	Participação sequencial por jogadores alternados.
Estados de vitória	Objetivos que tornam um jogador ou grupo o vencedor.

Fonte: Werbach e Hunter (2012); Wood e Reiners (2015)

A Tabela 3 apresenta as descrições mais importantes nas mecânicas de jogos. Segundo Werbach e Hunter (2012), as mecânicas são o processo básico que impulsiona a ação e gera engajamento entre os jogadores. Os desafios, por exemplo, podem ser descritos como uma lista de objetivos que exigem esforço do usuário para que sejam concluídos. O acaso atribui aos usuários uma sensação de incerteza e diversão, uma forma de evitar o tédio nas atividades propostas já que algo totalmente inesperado e aleatório poderá acontecer (WOOD; REINERS, 2015).

A competição promove o engajamento dos usuários nas atividades, e a cooperação auxilia na interação entre eles para que busquem alcançar um objetivo em comum. O *feedback* é bastante útil para fornecer informações sobre desempenho e progresso dos usuários através de tabelas de classificação, mensagens ou outras exibições visuais e

informativas. Já as aquisições de recursos ou recompensas tornam o jogo mais interessantes aos usuários, pois os motivam a realizar ações com o intuito de receberem algum benefício ao final (WERBACH; HUNTER, 2012).

As transações entre usuários permitem a negociação de recursos, bens virtuais e promovem a interação. Os turnos correspondem a uma estratégia em que os usuários podem revezar durante as atividades, cada usuário decide suas ações e as realiza durante seu turno (WERBACH; HUNTER, 2012). Por fim, têm-se os estados de vitória em que toda vez que o usuário realizar uma ação desejada, poderá receber algum tipo de recompensa (WOOD; REINERS, 2015).

Segundo Costa e Marchiori (2015) o nível de componentes é o mais concreto dos elementos de jogos usados na gamificação, pois corresponde às aplicações específicas visualizadas e utilizadas na interface do jogo. Ainda segundo os autores, assim como uma mecânica se liga com uma ou mais dinâmicas, vários componentes podem fazer parte de uma mecânica. Podem ser identificados alguns componentes de jogos, apresentados na Tabela 4.

Tabela 4 - Componentes de Jogos

COMPONENTES	DESCRIÇÃO
Pontos	Representações numéricas da progressão do jogo
Emblemas	Representações visuais de conquistas.
Quadro de líderes	Exibições visuais da progressão e realização do jogador.
Desbloqueio de Conteúdo	Aspectos disponíveis somente quando os jogadores atingem os objetivos
Níveis	Passos na progressão do jogador
Missões	Desafios prévios com objetivos e recompensas
Equipes	Grupos de jogadores trabalhando juntos por um objetivo comum
Avatares	Representações visuais de um jogador.

Bens Virtuais	Ativos de jogos com valor percebido ou dinheiro real

Fonte: Werbach e Hunter (2012); Wood e Reiners (2015)

A Tabela 4 apresenta as descrições dos componentes de jogos. Segundo Werbach e Hunter (2012) componentes são formas mais específicas que a mecânica ou a dinâmica podem suportar. Os pontos e os emblemas são fundamentais para medir e fornecer um registro de sucesso ao cumprimento das atividades propostas aos usuários. Os possíveis emblemas a serem conquistados são geralmente apresentados aos usuários antecipadamente, como uma forma de motivá-los a atingir os objetivos propostos.

Os quadros de líderes auxiliam na exibição da progressão dos usuários e o sucesso relativo em comparação com os oponentes. Os níveis são identificadores do progresso do usuário ao longo do tempo, geralmente com base nos pontos obtidos. As missões são caracterizadas como atividades voltadas a um objetivo definido, em que o usuário deverá cumpri-las com a intenção de ser recompensado. As missões são definidas por objetivos (por exemplo, pontos de referência ou marcos) que devem ser precisos, compreensíveis e concisos (WOOD; REINERS, 2015).

Por fim, têm-se as equipes (ou times) que correspondem a grupos de usuários que trabalham juntos para alcançar um objetivo em comum, os avatares que representam os usuários visualmente no mundo virtual, e os bens virtuais, que são ativos percebidos pelos usuários como valiosos, ou servem para distinguir o usuário de alguma forma, proporcionando um senso de individualidade (WOOD; REINERS, 2015). Conforme Werbach e Hunter (2012), a seleção de componentes está relacionada à intenção e propósito do sistema, ao público-alvo e as ferramentas envolvidas para o desenvolvimento da aplicação.

Vale ressaltar que mesmo sabendo da importância dos elementos de jogos que compõem as categorias aplicáveis à gamificação, não se faz necessário o uso de todos os elementos na composição de um jogo (KUUTTI, 2013). Alguns são mais adequados e úteis do que outros, e podem não ser igualmente relevantes ao que será proposto em um determinado contexto. Dessa forma, é sugerida a utilização de elementos genéricos como base para criar sistemas gamificados e/ou servir como ponto de partida para o *designer* de jogo.

2.4 PLATAFORMAS DE GAMIFICAÇÃO

Com a popularização da gamificação, surgiu a necessidade de desenvolver um ambiente que possa evoluir e operar em cenários distintos. Dois trabalhos sobre arquiteturas de plataformas gamificadas serão apresentadas a seguir, que em linhas gerais propuseram uma forma de analisar o desempenho dos jogadores e compartilhar componentes e eventos entre toda a plataforma.

Para Sripada, Reddy e Khandelwal (2016), utilizar a abordagem da engenharia de linha de produtos facilita a implementação de uma arquitetura genérica que maximize a reutilização de componentes ao construir uma série de aplicativos. A abordagem de construção consiste em três fases:

1. **Gerenciamento de produto**

   Responsável por coletar e analisar todas as características, funcionalidades ou recursos desejados de sistemas de software, domínio ou componente.

2. **Engenharia de domínio**

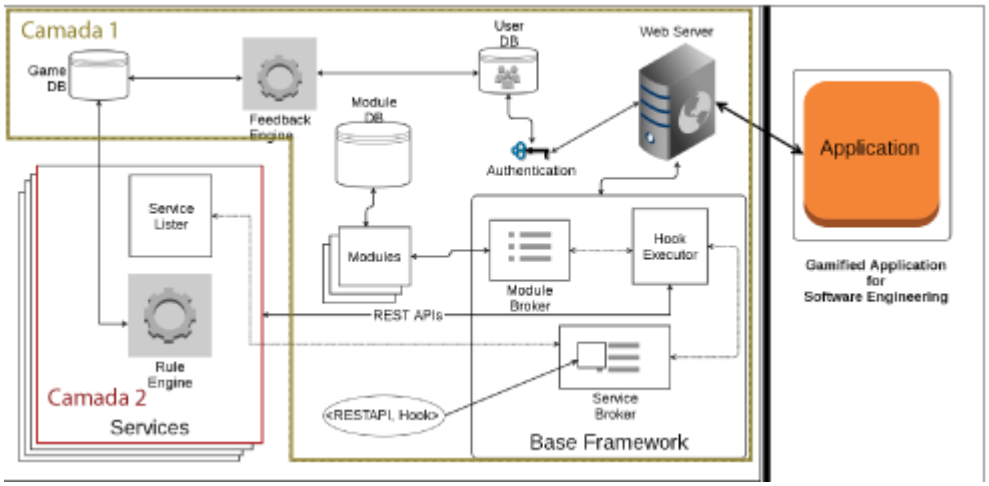
   Classificação e organização de partes de sistemas de software de acordo com sua funcionalidade. Esta organização é chamada de domínio. Seu principal objetivo é construir sistemas ou componentes no mesmo domínio, sob a forma de ativos reutilizáveis, com a finalidade de compor novos produtos em menos tempo e com menor custo.

3. **Engenharia de aplicação**

   Responsável por alcançar a maior realização possível dos ativos de domínio ao definir o desenvolvimento de uma aplicação, documentação dos artefatos de domínio e determinar os impactos das diferenças entre o aplicativo e o domínio.

Com a adoção de uma abordagem específica e o delineamento de suas fases, Sripada, Reddy e Khandelwal (2016) propuseram a arquitetura apresentada a seguir:

Figura 5 - Diagrama de arquitetura



Fonte: Sripada, Reddy e Khandelwal (2016)

Para os autores, Sripada, Reddy e Khandelwal (2016), a implementação da arquitetura de serviços com estilo REST facilita a segmentação dos elementos estáticos e dinâmicos do *design* de jogo. Conforme a Figura 5 apresenta, a arquitetura é dividida em duas camadas. A primeira compõe os componentes comuns, juntamente com banco de dados e serviços globais essenciais para criar um aplicativo gamificado. A segunda dispõe dos serviços necessários para fornecer todos os componentes de *design* de jogos para qualquer instância única de gamificação.

As duas camadas são compostas por módulos altamente desacoplados, possibilitando de forma fácil a inclusão de novos componentes, conforme necessário. Os principais módulos apresentados na Figura 5 são (SRIPADA; REDDY; KHANDELWAL, 2016):

- Módulo de autenticação (Authentication module): responsável por autenticar todas as solicitações de usuário antes de executar qualquer interação com o aplicativo;
- Módulo base (Base framework): contém os componentes comuns de *design* de jogos indicados na fase de engenharia de domínio. Além de disponibilizar dinamicamente para consumo de outros módulos e serviços os elementos de interface do usuário, como gráficos, tabelas e formulários. O que propicia a uma economia de tempo e esforço ao desenvolvedor;
- Módulos (Modules): conjunto de componentes utilitários que executam atividade de rastreamento de erros, revisão de código, documentação, entre outras. Também é responsável por registrar no banco de dados dos módulos

as URIs, eventos e funcionalidades dos módulos, sempre que um novo módulo é incluso no módulo base;

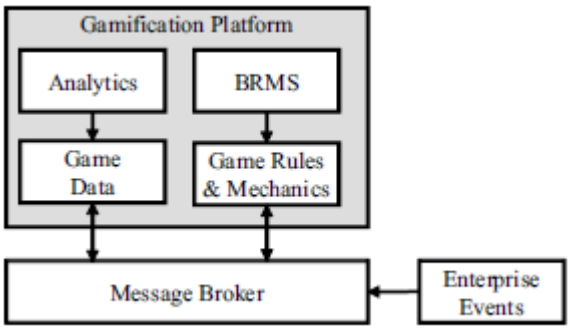
- Agente de módulo (Module Broker): disponibiliza uma API responsável por carregar e interagir com módulos;
- Serviço e lista de serviço (Services and Service Lister): os serviços disponibilizam APIs REST que são responsáveis por apresentar os elementos de *design* do jogo modelados como serviços. A lista de serviços fica responsável por realizar a comunicação com o módulo base para registrar as URIs da API REST do serviço;
- Agente de Serviço (Service Broker): responsável por realizar a comunicação entre o módulo base e a lista de serviços. O agente de serviço detém o <REST API, Hook>, um mapa de lista de serviços, que é atualizado automaticamente, após a inclusão de um novo serviço no módulo base;
- Executor de eventos (Hook Executor): a cada ação do usuário, um novo evento é transmitido, ficando a cargo do executor de eventos executar todas as tarefas atribuídas ao evento. Além de consumir os serviços que interagem com o evento específico invocando a consulta no mapa <REST API, Hook>;
- Regras de negócio (Rule Engine): dependendo da interação que o usuário faz no jogo por meio de atividades ou eventos, a regras de negócio fica responsável por recompensar o usuário e alterar os elementos de *design* do jogo, conforme as regras e regulamentos definidos pelo desenvolvedor.
- Sistema de feedback (Feedback Engine): para avaliar o desempenho do usuário ao decorrer do jogo, torna-se necessário criar um mecanismo que possa coletar os dados das atividades ou eventos que produzam recompensas aos usuários. Esse mecanismo fica a cargo do sistema de feedback que a todo momento, em segundo plano, captura dos dados das regras de negócio.
- Bancos de dados (Databases): os bancos de dados são responsáveis por armazenar os dados do jogo, dados do módulo e dados do usuário.

Os elementos de *design* de jogos são servidos como Serviços Web REST, com cada serviço sendo atribuído a uma URI exclusiva. Assim, é possível minimizar o tempo do desenvolvedor ao implementar um novo elemento de *design*, por exemplo, ao incluir um conjunto de emblemas que será enviado ao usuário conforme sua pontuação aumenta.

Primeiramente os emblemas são armazenados no servidor web (Web Server) em formato de arquivos de imagem e renomeado para que cada arquivo tenha um nome único. Os *metadados* dos arquivos e suas respectivas pontuações são armazenadas no banco de dados do jogo (Game DB) para ser consumido pelos serviços das regras de negócio (Rule Engine). Por último, o desenvolvedor deve definir as regras de negócio (Rule Engine) e quais eventos <REST API, Hook> o usuário terá que realizar para receber o emblema e sua pontuação (SRIPADA; REDDY; KHANDELWAL, 2016).

Na Figura 6 é apresentada a arquitetura de plataforma gamificada para sistemas corporativos desenvolvida por Herzig, Ameling e Schill (2012).

Figura 6 - Plataforma gamificada



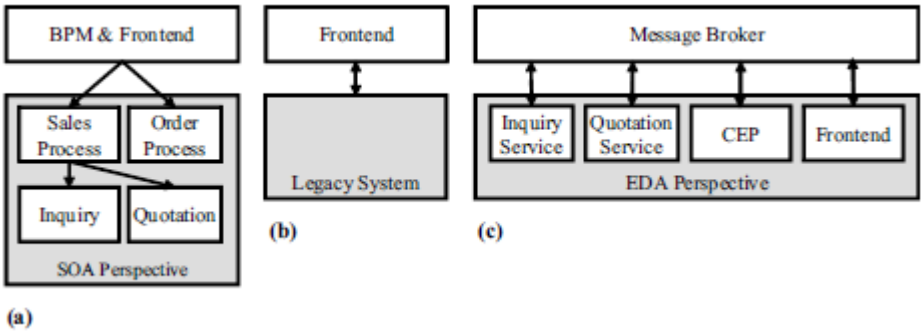
Fonte: Herzig, Ameling e Schill (2012)

Conforme a Figura 6 demonstra, tudo que é importante para a gamificação pode ser representado por eventos (Enterprise Events). Para exemplificar essa arquitetura, os autores apresentam o seguinte exemplo: o usuário A completa a etapa T no processo P com sucesso. Para esta ação existe um evento E que recebe todos os parâmetros e, assim, o evento E = (A, T, P). O evento é enviado para o mecanismo que processa os eventos de acordo com as regras de negócio do jogo (Game Rules & Mechanics). As regras são criadas e gerenciadas por meio do sistema de gerenciamento de regras de negócios BRMS.

Antes de processar o evento, o mecanismo (Game Rules & Mechanics) avalia cada evento de acordo as regras do jogo definidas no gerenciamento de regras BRMS. Cada regra é uma condição se-então $X \Rightarrow Y$, onde o resultado Y é executado imediatamente quando a premissa X é verdadeira. Quando o usuário receber uma recompensa por um determinado processo, fica a cargo do mecanismo enviar o evento de recompensa para esse usuário por meio do agente de mensagens (Message Broker). O evento contendo a recompensa segue para o repositório do jogo (Game Data), onde é processado e armazenado no banco de dados. Por fim, um componente de análise de comportamento do jogador é usado para analisar os

dados armazenado no banco, a fim de otimizar as regras do jogo avaliando o engajamento do jogador (HERZIG; AMELING; SCHILL, 2012).

Figura 7 - Arquitetura proposta



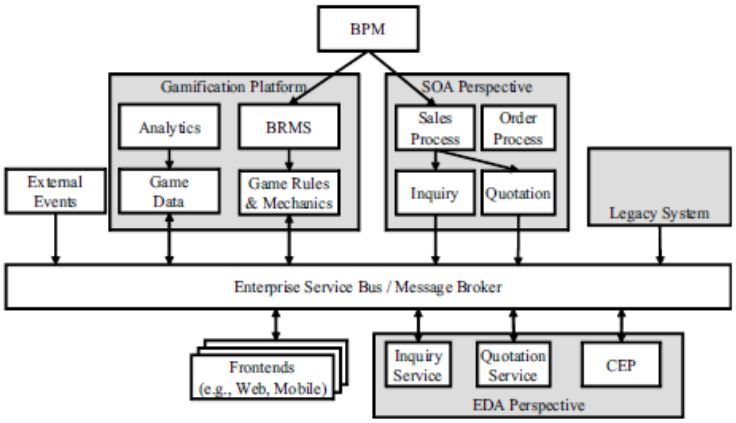
Fonte: Herzig, Ameling e Schill (2012)

A Figura 7 apresenta as combinações de sistemas que pertencem ao contexto geral da plataforma. Os grupos de sistemas são (HERZIG; AMELING; SCHILL, 2012):

a. uma SOA em que os serviços são gerenciados pelo sistema de gerenciamento de processos de negócios BPM e enviados ao usuário;
b. um sistema legado formado por funcionalidades complexas ligado diretamente ao usuário através do *front-end*;
c. uma arquitetura orientada a eventos onde o CEP gerencia a interação entre os serviços e usuário.

Por fim, a plataforma é conectada a uma ferramenta de análise que fornece suporte à tomada de decisão, a *Enterprise Information System* (EIS). A conexão é feita por intermédio do agente de mensagens (Message Broker) ou do serviço corporativo ESB, conforme apresentado na Figura 8.

Figura 8 - Arquitetura da plataforma gamificada



Fonte: Herzig, Ameling e Schill (2012)

A plataforma recebe eventos externos enviados pelos *front-end* dos respectivos EIS que são capturados pelo ESB. Os *front-end* podem ser área de trabalho, clientes web ou aplicações móveis responsáveis por capturar em suas interfaces os eventos dos usuários. Além de enviar eventos, os *front-end* também recebem, por exemplo, quando o usuário atinge um novo nível, o mecanismo do jogo envia o evento de recompensa para o *front-end*, que exibe uma notificação ao usuário. Além disso, a plataforma vincula o BPM ao BRMS, que gera a facilidade de o usuário corporativo alterar as regras do jogo com uma maior agilidade e flexibilidade (HERZIG; AMELING; SCHILL, 2012).

A arquitetura proposta por Sripada, Reddy e Khandelwal (2016) mostra-se importante para o desenvolvimento do presente trabalho. Seguindo o modelo arquitetural, será possível implementar a plataforma gamificada em camadas composta por módulos altamente desacoplados, facilitando a inclusão de novos módulos ou componentes, conforme necessário.

O trabalho de Herzig, Ameling e Schill (2012) estabelece que a arquitetura da plataforma pode ser baseada em eventos. Esses eventos são interações que o usuário realiza na interface e são processadas por mecanismos conforme as regras de negócios.

Tanto o trabalho de Sripada, Reddy e Khandelwal (2016) como o de Herzig, Ameling e Schill (2012) apresentam uma proposta de arquitetura de forma satisfatória para desenvolver plataformas gamificadas. Entretanto, levando em consideração os objetivos do presente trabalho, a arquitetura apresentada por Herzig, Ameling e Schill (2012) é mais adequada para ser empregada nas resoluções dos objetivos, devido a sua estrutura estabelecida em camadas, divisões mais específicas dos componentes e facilidade para integrar novos módulos, além de dividir os bancos de dados, facilitando a análise de dados.

3 MATERIAIS E MÉTODOS

Esta seção apresenta os materiais que utilizados e a metodologia aplicada para o desenvolvimento da plataforma.

3.1. MATERIAIS

Para o desenvolvimento da plataforma, foram utilizados os seguintes materiais;

- Node.js;
- AdonisJS;
- AngularJS;
- Swagger;
- MySQL;

Como tecnologia de *back-end*, foi empregado o *Node.js*. Um interpretador de código *JavaScript* que é executado sobre a máquina virtual *JavaScript* V8 *Engine* da Google. Sua grande vantagem é a possibilidade de construir aplicações rápidas e escaláveis, sendo ideal para aplicações em tempo real, com intenso tráfego de dados entre vários dispositivos distintos (NODEBR, 2019).

O *framework* modular *AdonisJS* torna possível agilizar o desenvolvimento da plataforma baseada em serviços, além de possibilitar a inclusão de novos módulos de forma simples. Sua estrutura em MVC para *Node.js* oferece vários provedores de serviços estáveis para desenvolvimento de aplicações Web no lado do servidor. Isso permite manter o foco na regra de negócio, sem perder muito tempo com funcionalidades que são nativas no *framework*, como manipulação de banco de dados, autenticação de usuário, envio de e-mail, serviços de tempo real, entre outros, assim desenvolver uma aplicação robusta e confiável leva menos tempo (ADONISJS, 2019).

Para acelerar o desenvolvimento no lado do cliente, foi utilizado o *framework AngularJS*, que dispões de uma estrutura para criação de aplicações Web dinâmicas no padrão MVC. Seshadri e Green (2014) afirmam que o "*framework* oferece uma estrutura consistente e escalável, que facilita desenvolver aplicações complexas e de grande porte como parte de uma equipe".

Com a finalidade de padronizar a documentação da plataforma, foi empregado a *framework Swagger* (Open RESTful API Representation Language). Seu objetivo busca a padronização de APIs REST com especificação e documentação de recursos, como

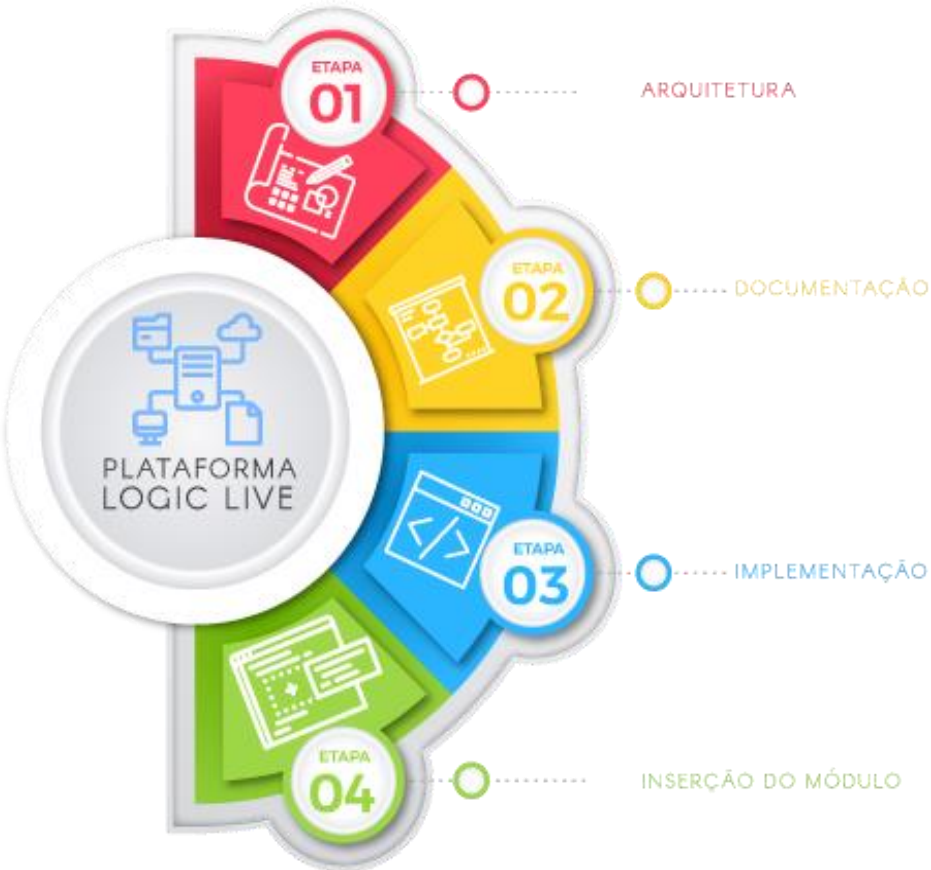
endpoints, parâmetro de entrada, objetos de retorno, código HTTP, métodos de autenticação, entre outros (NASCIMENTO, 2018). Atualmente grandes empresas como Google, Microsoft e IBM, apoiam o projeto *Swagger* (LIMA, 2016).

Por fim, optou-se por utilizar o banco de dados *MySQL*. Milani (2006, p. 21) afirma em seu livro que o "*Mysql* é um banco de dados completo, robusto e extremamente rápido, com todas as características dos principais bancos de dados pagos existentes no mercado". No início o *MySQ*L foi desenvolvido para atender aplicações pequenas e médias, mas com as atualizações de versões, a sua capacidade de armazenamento foi sendo expandida, atualmente é possível armazenar até 65.536 TB (terabytes) por tabela do tipo *InnoDB*. Na atualidade, é banco de dados mais utilizado em aplicações Web, e de soluções que requer alta demanda e segurança nas transações, tais como lojas virtuais (MILANI, 2006).

3.2. MÉTODOS

Os métodos para o desenvolvimento da plataforma gamificada foram divididos em 4 etapas, listadas a seguir.

Figura 9 - Etapas de desenvolvimento do trabalho



A Figura 9 apresenta as etapas realizadas para o desenvolvimento da plataforma. Para desenvolver a plataforma gamificada, os seguintes procedimentos foram realizados:

1 Arquitetura

A primeira etapa deste trabalho consistiu em reuniões com o especialista de domínio, a fim de decidir quais os conjuntos de requisitos necessário para desenvolver a plataforma. Além de classificar e organizar os requisitos comuns em domínio e camadas para possibilitar a implementação de componentes reutilizáveis com mais facilidade viabilizando a elaboração de uma estrutura arquitetônica orientada a serviços, organizando os componentes comuns, e seus relacionamentos em conjunto de serviços ou recursos independentes com baixo acoplamento. Para possibilitar que fosse empregada a arquitetura seguindo o modelo proposto por Sripada, Reddy e Khandelwal (2016), assim foi possível atender a todos os requisitos técnicos e operacionais da plataforma, otimizando o desempenho, capacidade de gerenciamento e segurança;

2 Documentação

Nesta etapa foi gerada a documentação utilizando o *framework Swagger* (apresentado na seção 4.3). A documentação torna-se necessária para que novos módulos possam integrar e interagir com a plataforma, sem a obrigação dos desenvolvedores conhecer o código implementado na plataforma. Com a documentação é possível entender especificações dos recursos REST, como: URI de acesso, parâmetros necessários, padrão de resposta, padrão de erro, tipo de dados, entre outros.

3 Implementação

A implementação da plataforma teve como base a arquitetura desenvolvida na primeira etapa. Os recursos foram agrupados em domínios e o processo de desenvolvimento foi implementado em dois estágios:

- *back-end*: implementação da API empregando o *framework* modular *AdonisJS*, para disponibilizar os recursos em padrão API REST por meio das URL e métodos do protocolo HTTP;
- *front-end*: desenvolvimento das interfaces do jogador e gestão, utilizando o *framework AngularJS* para exibir e processar as páginas, além de consumir os recursos disponibilizado pela API REST.

4 Inserção de módulos

A última etapa do presente trabalho foi a inserção do módulo Tabela Verdade, seguindo a documentação gerada na etapa anterior, além de disponibilizar a plataforma e documentação para integração de novos módulos por meio da internet, a fim de que outros jogadores possam interagir com os módulos e novos desenvolvedores sejam capazes de incluir módulos seguindo a documentação da plataforma.

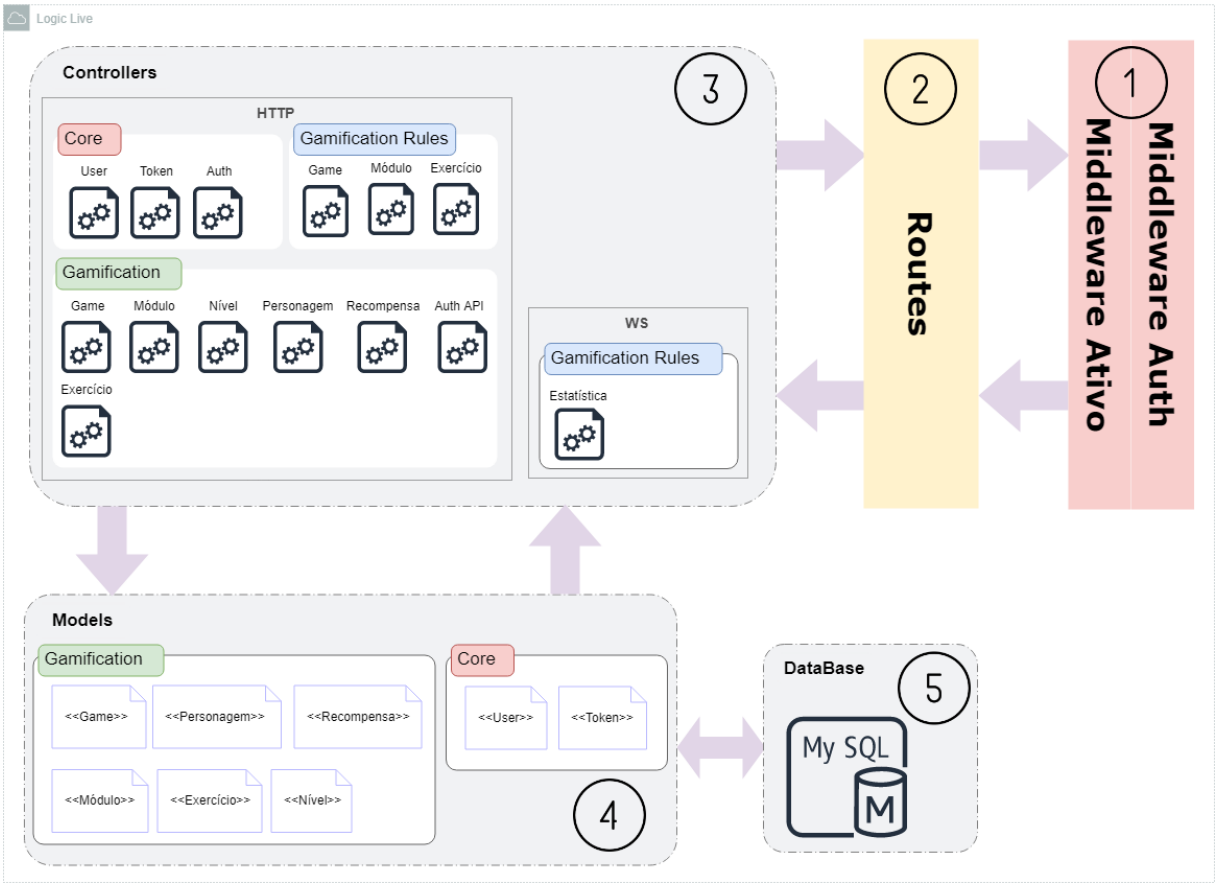
4 RESULTADOS

Esta seção apresenta o modelo de plataforma desenvolvido neste trabalho, com a descrição da sua arquitetura, documentação e processo de inserção do módulo de Tabela Verdade.

4.1 ARQUITETURA

A estrutura da plataforma foi classificada e organizada em três partes altamente desacopladas de acordo com sua funcionalidade. Esta classificação é chamada de domínio, e a representação da arquitetura pode ser visualizada na Figura 10.

Figura 10 - Arquitetura API Logic Live



Os domínios apresentados na Figura 10 são:

- Core: contém os componentes comuns relativos aos usuários da plataforma, além de disponibilizar dinamicamente para consumo de outros componentes e serviços o gerenciamento e autenticação dos usuários. Isso proporciona ao desenvolvedor uma

economia de tempo e esforço ao criar novas funcionalidades que dependem de autenticação;

- Gamification: abrange os componentes comuns da gamificação. Responsável por disponibilizar dinamicamente para consumo de outros módulos e serviços os elementos de jogos, como exercícios, personagem, recompensas entre outros;
- Gamification Rules: dependendo da interação que o usuário faz no jogo por meio de atividades ou eventos, as regras da gamificação ficam responsáveis por recompensar o usuário e alterar os elementos de *design* do jogo, conforme as regras e regulamentos definidos pelo desenvolvedor no domínio *Gamification*.

Além de utilizar o domínio para classificar e organizar as funcionalidades, também foi empregado a divisão por camadas. Esta divisão proporciona mais flexibilidade e oportunidades de reuso dos recursos. As camadas apresentadas na Figura 10 serão detalhadas nas subseções a seguir.

4.1.1 Middleware

São funções que foram desenvolvidas para serem executadas em sequências que permitem filtrar solicitações HTTP e WS antes de alcançar as rotas.

- Auth: verifica se o usuário está autenticado. Se o usuário não estiver autenticado ou seu *token* for inválido, uma exceção é lançada antes da requisição chegar ao manipulador de rotas.
- Ativo: verifica se o usuário está ativo. Seu principal objetivo é evitar a repetição de código para verificar se um determinado usuário está ativo nas classes do *controllers*, assim a verificação é feita antes da requisição chegar em qualquer classe. Se o usuário não estiver ativo, uma exceção com uma mensagem é enviada.

4.1.2 Routes

Conjunto de identificadores únicos, empregados como interface para acesso aos recursos por

meio de URLs.

Os identificadores foram divididos em 3 grupos:

- API: contém todas as URLs destinadas a fornecer recursos para integração de novos módulos;
- Gamification: grupo responsável por disponibilizar os recursos para controle da plataforma. Como listagem de games ou exercícios;
- Gestão: grupo encarregado de oferecer os recursos de análise do perfil do jogador, gerenciamento de módulos e desenvolvedores, entre outras funcionalidades necessárias para gestão da plataforma.

4.1.3 Controllers

Repositório de classes responsável por manipular as requisições seguindo as regras de negócio da plataforma. Os *controllers* foram divididos em dois protocolos: HTTP e WS.

Protocolo HTTP é responsável por atender uma requisição e realizar uma resposta na mesma

requisição. O cliente faz uma solicitação completa, o servidor fornece uma resposta completa

e a conexão é fechada. Os domínios e classes são:

- Core: domínio de uso geral dividido em três classes.
    - User: classe responsável por retornar informações sobre o jogador no ambiente da plataforma;
    - Token: responsável por criar, atualizar e deletar o *token* do jogador;
    - Auth: a classe fica responsável pelo *login* e *logout* do jogador na plataforma.
- Gamification Rules: domínio dividido em três classes, utilizado para capturar atividades ou eventos e alterar os elementos de *design* do jogo, conforme as regras de negócio.
    - Game: retorna as informações sobre o progresso do jogador nos módulos da plataforma;
    - Módulo: responsável por retornar os recursos de um único módulo, por exemplo, as informações de níveis do Módulo Tabela Verdade;
    - Exercício: classe responsável por disponibilizar os recursos necessários para receber a resposta do jogador ao exercício.

- Gamification: domínio utilizado somente pelo desenvolver do módulo. O jogador não tem acesso a este domínio.
    - Game: classe responsável por validar e realizar a manipulação das informações sobre o módulo enviado pelo desenvolvedor através da API REST;
    - Recompensa: gerencia os dados das recompensas cadastradas pelo desenvolvedor;
    - Personagem: faz o gerenciamento dos personagens cadastrado no módulo;
    - Módulo: apesar da semelhança do nome, o módulo não deve ser confundido com o módulo manipulado pela classe game. Este módulo é responsável por gerenciar o agrupamento dos níveis;
    - Nível: classe que cuida da gerência dos níveis;
    - Exercício: responsável por disponibilizar os recursos necessário que os desenvolvedores possam cadastrar, alterar e excluir os exercícios.
    - Auth API: classe que dispõe os recursos de autenticação e manipulação de dados dos desenvolvedores, como dados de cadastro e geração ou revogação de *token*.

Protocolo WS é utilizado para receber informações em tempo real sem a interação do cliente. As conexões WS são bidirecionais, *full-duplex* e de longa duração. O cliente ou servidor podem enviar mensagens assíncronas a qualquer momento, sem a necessidade de acessar uma URL. Os domínios e classes são:

- Gamification Rules: classe responsável por disponibilizar recursos de captura de eventos ou atividades dos jogadores.
    - Estatísticas: responsável por atualizar em tempo real as informações de *ranking* de todos os jogadores conectados por meio de *broadcast.* A classe também captura a informação enviada pelo módulo quando o jogador responde a um exercício. Se a resposta for correta a classe envia uma mensagem e a URL do próximo exercício ao jogador que respondeu, caso contrário uma mensagem de erro é enviada.

4.1.4 Models

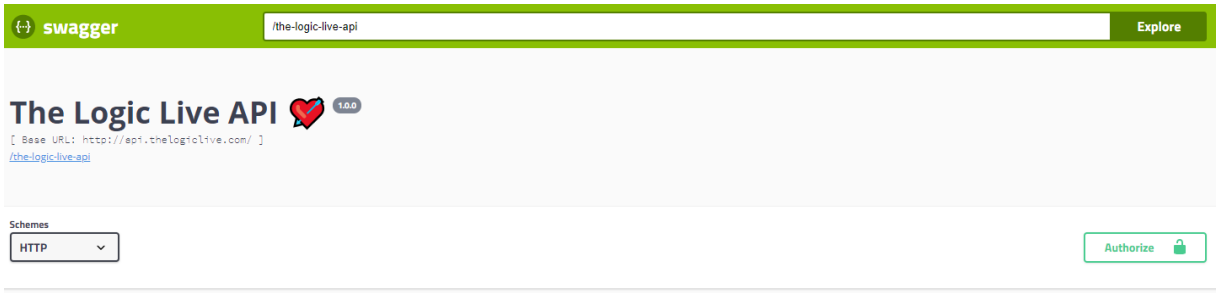
Responsável por realizar todas as operações de consultas, inserção e atualização de dados no banco de dados e retornar o valor para o *controllers*. Dividir em dois domínios foi fundamental para a organização no banco de dados das funcionalidades da plataforma. Esta divisão possibilitou, por exemplo, utilizar o *Core* para para servir outra plataforma que utilize autenticação de usuários.

- Core: domínio para uso geral dividido em duas classes:
    - User: classe responsável por manipular e enviar ao banco de dados as informações dos usuários;
    - Token: responsável por gerenciar o *token* de cada usuário.
- Gamification: domínio responsável por conter as classes de manipulação ao banco de dados referente a gamificação.
    - Game: responsável por manipular no banco dados os recursos disponíveis aos desenvolvedores para cadastro e alteração dos módulos, por exemplo o módulo de Tabela Verdade;
    - Recompensa: realiza a leitura, escrita e validação dos dados referente às recompensas no banco de dados, conforme são solicitadas pelos recursos;
    - Personagem: responsável por manipular as solicitações de leitura e escrita ao banco de dados das informações dos personagens, como código, nome e imagem;
    - Módulo: classe que engloba as funcionalidades de criação, consulta, atualização e exclusão (CRUD) dos dados da tabela módulo. A classe opera como um agrupador de Níveis, contendo as informações do Game, Recompensa e Personagem;
    - Nível: responsável por gerenciar os dados dos níveis e enviar ao banco de dados;
    - Exercício: principal classe do domínio *Gamification* no model, é responsável por realizar o CRUD das informações pertinente aos exercícios.

4.2 DOCUMENTAÇÃO

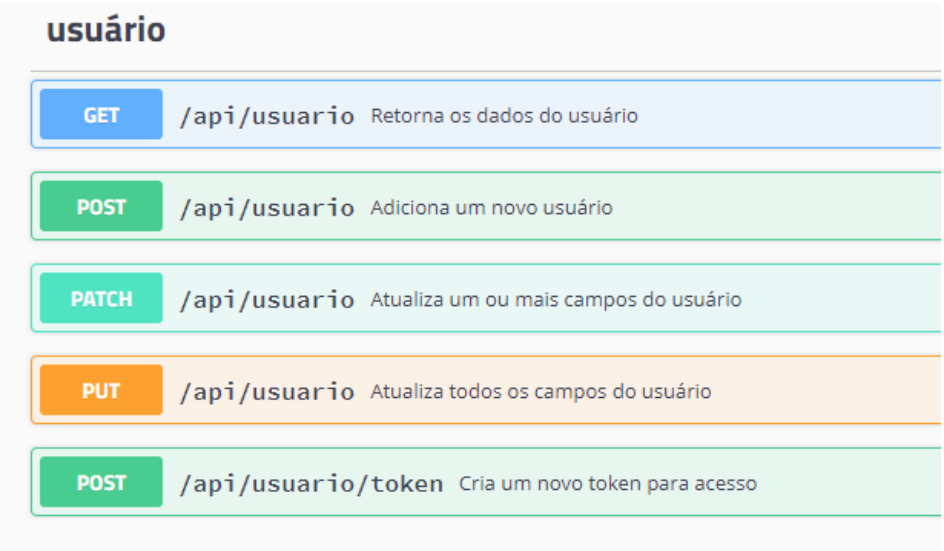
Com o *framework Swagger* foi possível gerar a documentação dos recursos da API de forma que os desenvolvedores possam entender e consumir os serviços sem a necessidade de conhecer o código implementado no servidor da plataforma.

Figura 11 - Documentação Logic Live



A figura 11 apresenta um *preview* dos recursos do Logic Live (Visualização completa em *http://api.thelogiclive.com/docs/*) utilizando a ferramenta *Swagger UI* que faz parte do *framework Swagger*. A ferramenta permite que os desenvolvedores tenham uma visão geral e clara de como a API retorna suas requisições e quais parâmetros e opções são necessárias para consumir cada recurso da plataforma, além de disponibilizar mecanismo de consumo para que os desenvolvedores consigam interagir e realizar testes sem a necessidade de desenvolver um cliente REST.

Figura 12 - Recurso do usuário



Como apresentado na Figura 12, foi utilizado a ferramenta *Swagger UI* na modelagem dos recursos que possibilitam a geração de um novo *token*, consulta, inclusão, alteração e

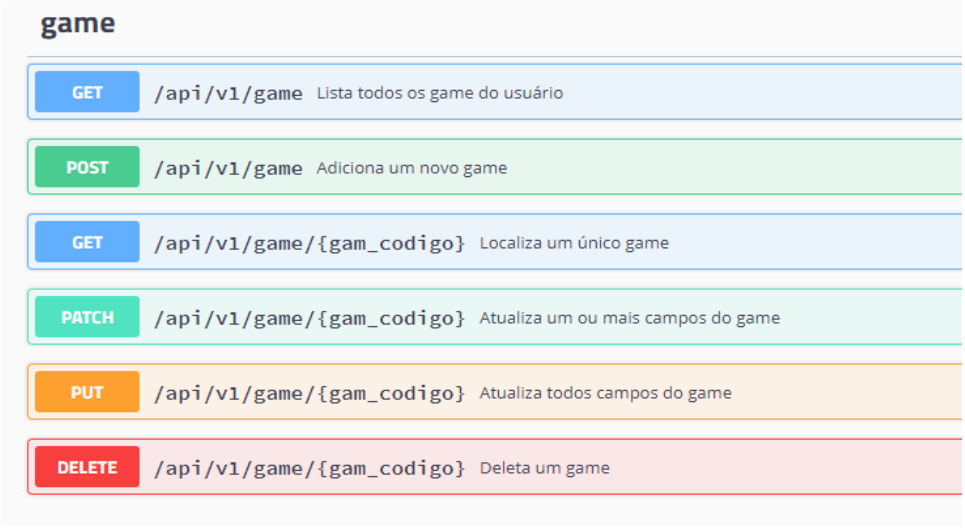
exclusão dos dados do desenvolvedor, que para facilitar o entendimento o recurso foi chamado de “usuário”.

A modelagem do recurso “usuário” é composta por 2 rotas em 4 diferentes tipos de métodos do protocolo HTTP. A rota */api/usuario* contém 4 métodos: GET, responsável por retornar os dados do desenvolvedor, como nome, nome de usuário, login, e-mail e *token*; POST, realiza a inclusão de um novo desenvolvedor, sendo necessário passar como parâmetro o nome, nome de usuário, e-mail e senha; PATCH, atualiza um ou mais dados do desenvolvedor, sendo que é essencial informar no parâmetro quais campos desejam ser atualizados; PUT, também utilizado para atualizar os dados, porém sua principal diferença do método PATCH é que são necessários informar todos os campos referente ao desenvolvedor, como informado no método POST. Já a rota */api/usuario/token* é composta por apenas um único método (POST), que requer como parâmetro o e-mail e senha do desenvolvedor para que o *token* existente seja refogado e o novo *token* gerado, impossibilitando que o *token* refogado tenha acesso aos recursos que demanda autenticação.

Os demais recursos foram modelados de forma a seguir um padrão de duas rotas e 5 métodos, com exceção ao recurso de respostas (Figura 14). Adicionando apenas o método DELETE, além dos já apresentado anteriormente. O método é responsável por requerer a exclusão de um dado. Na Figura 13, é possível verificar a modelagem do método DELETE para excluir um determinado GAME.

A plataforma visa a inclusão de vários módulos para ensino de lógica de predicados, como o módulo Tabela Verdade. Para que fosse possível a integração dos módulos na plataforma, o recurso GAME foi implementado.

Figura 13 - Recurso do game



A Figura 13 apresenta a modelagem dos recursos do GAME, seguindo o padrão de duas rotas com 5 métodos (GET, POST, PATCH, PUT e DELETE), a modelagem dos recursos possibilitam consultar, incluir, atualizar e excluir um determinado módulo da plataforma.

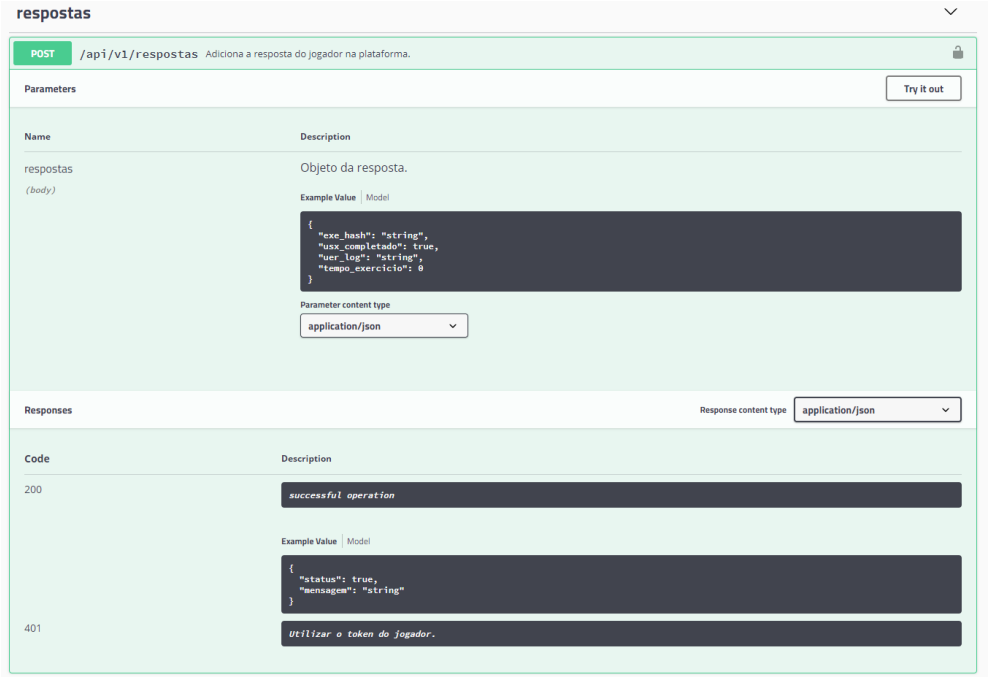
Na primeira rota, */api/v1/game*, apenas dois métodos são utilizados: GET e POST. Sem a necessidade de informar parâmetros, o GET lista todos os módulos cadastrados na plataforma pelo desenvolvedor que deve informar o *token* para consumir o recurso. Já o POST realiza o cadastro do módulo, tendo como obrigação informar nos parâmetros o nome do módulo, descrição e se está ativo. Ao consumir o recurso com o método POST, os dados do módulo são retornados, entre eles o código do módulo, que não foi informado pelo desenvolvedor, já que sua geração é de responsabilidade da plataforma.

Para consumir os recursos da segunda rota, */api/v1/game/{game_codigo}*, é indispensável que o desenvolvedor informe o código do módulo no lugar de *{game_codigo}*, assim ao consumir, por exemplo, a rota com o método GET, apenas os dados do módulo informado serão retornados, não sendo necessário listar os dados de todos os módulo, conforme ocorre na primeira rota.

A modelagem dos recursos do GAME adota o estilo API REST que utiliza uma única URL para disponibilizar os recursos. O mesmo padrão é adotado em RECOMPENSA, PERSONAGEM, MÓDULO, NÍVEL e EXERCÍCIO, disponíveis em *http://api.thelogiclive.com/docs/*. O padrão também especifica os parâmetros e as informações de retorno de cada recurso. Para visualização desses parâmetros e retornos a

ferramenta *Swagger UI*, dispõe de janelas (Figura 14) que possibilitam os desenvolver visualizar de forma mais fácil os dados de envio e retorno.

Figura 14 – Recurso de respostas

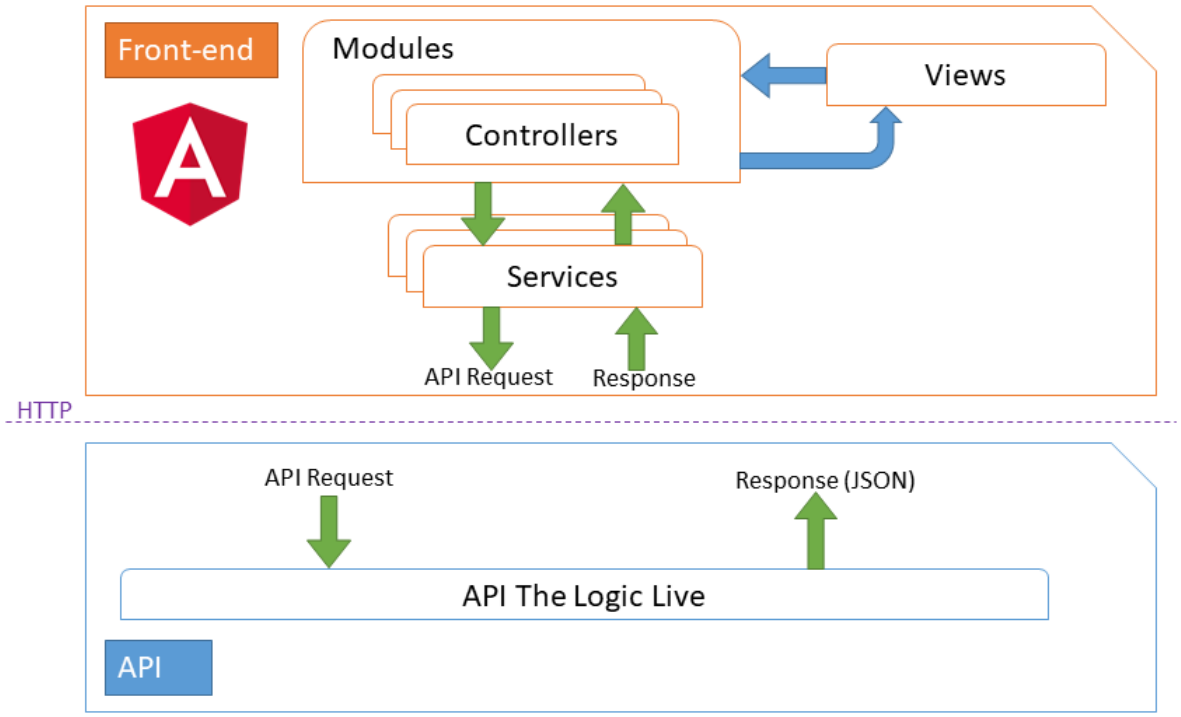


Ao clicar na URL do recurso, uma aba com as informações detalhadas sobre os parâmetros de envio e dados de resposta é exibida, conforme apresenta a Figura 14. Em *Parameters* são apresentados quais campos e tipos de dados a solicitação requer, sendo eles obrigatórios ou não. As possíveis respostas são apresentadas em *Responses*, seguindo o padrão API REST, onde cada tipo de resposta tem um código HTTP específico.

4.3 IMPLEMENTAÇÃO

Esta seção apresenta a forma com a plataforma foi desenvolvida, além de apresentar as páginas gerais do jogador e gestão da plataforma.

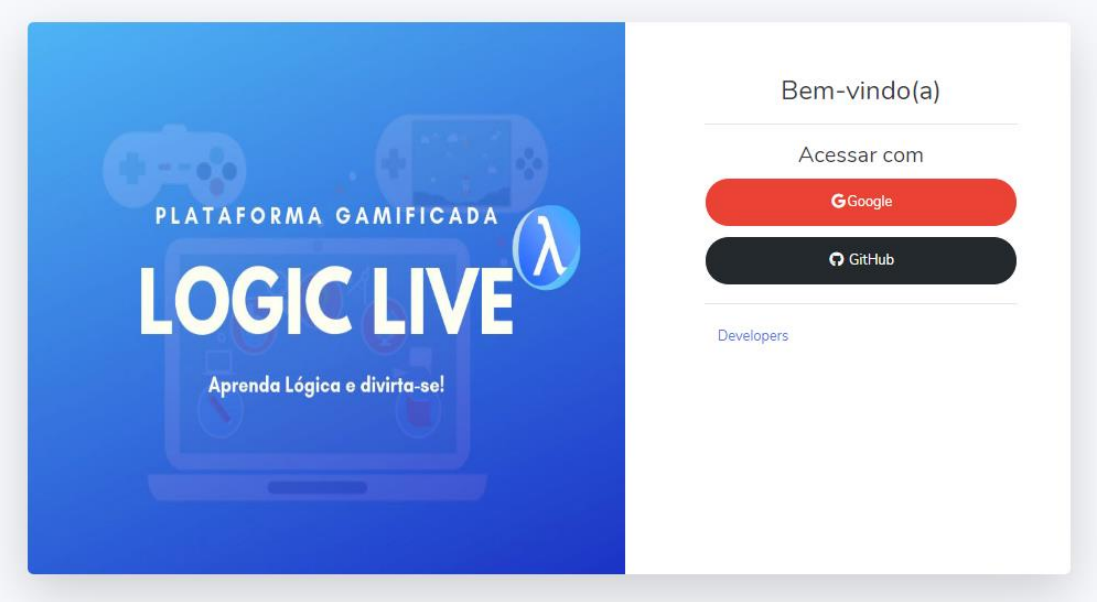
Figura 15 - Arquitetura front-end



A implementação da interface do jogador e gestão segue a arquitetura apresentada na Figura 15. A comunicação com a API é feita por meio dos métodos do protocolo HTTP, utilizando o padrão de API REST, conforme a documentação apresentada na subseção 4.2. Com essa estrutura foi possível ter um ganho de tempo no desenvolvimento, visto que o foco do desenvolvimento ficou exclusivamente no *front-end*, não sendo necessário criar os recursos no servidor, já que a API dispõem dos recursos, bastando apenas consumi-los.

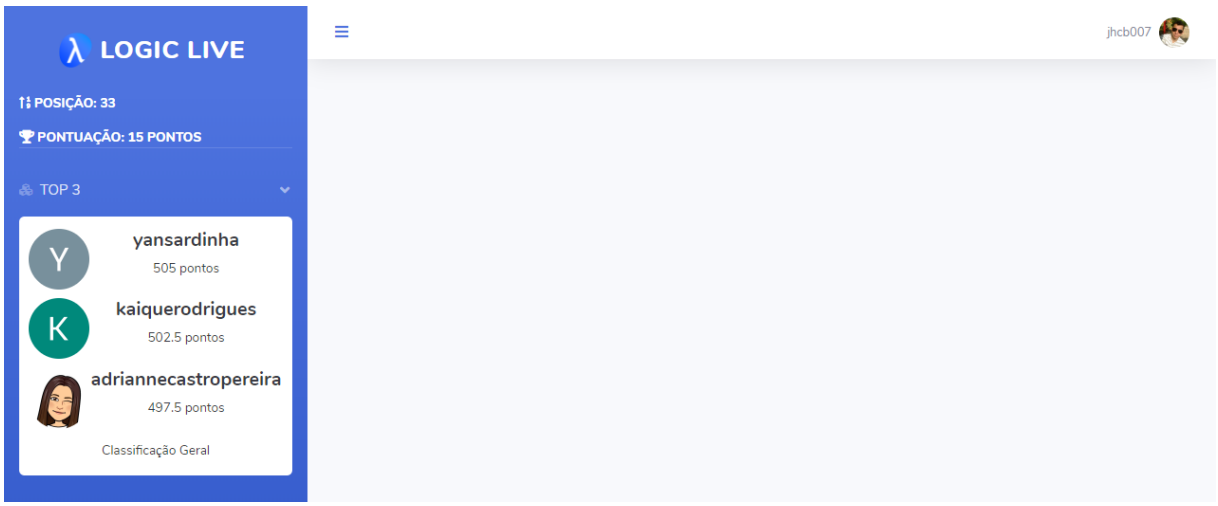
O *framework AngularJS* é responsável pelas atualizações dos dados, que através dos *services*, realizam requisições enviando dados no formato JSON para API. Após processar a solicitação, a API retorna o JSON que é recebido pelos *services*, processados nos *controllers* e, por fim, exibidos na página, sem a necessidade de atualizar toda a página ou realizar uma nova requisição nas *views* (HTML).

Figura 16 - Página de acesso



A Figura 16 apresenta a página de acesso implementada para a plataforma. Nesta página optou-se por utilizar o social login, tendo em vista que são poucos os usuários da internet que não têm contas em pelo menos uma rede social. Dessa forma, o jogador não precisa preencher um formulário de cadastro nem memorizar um nome de usuário ou senha. Após clicar no botão que representa a sua rede social, o usuário é redirecionado para página inicial do módulo (Figura 17). Por último, o *link* "Developers" redireciona para página de documentação da API (Figura 11).

Figura 17 - Página inicial

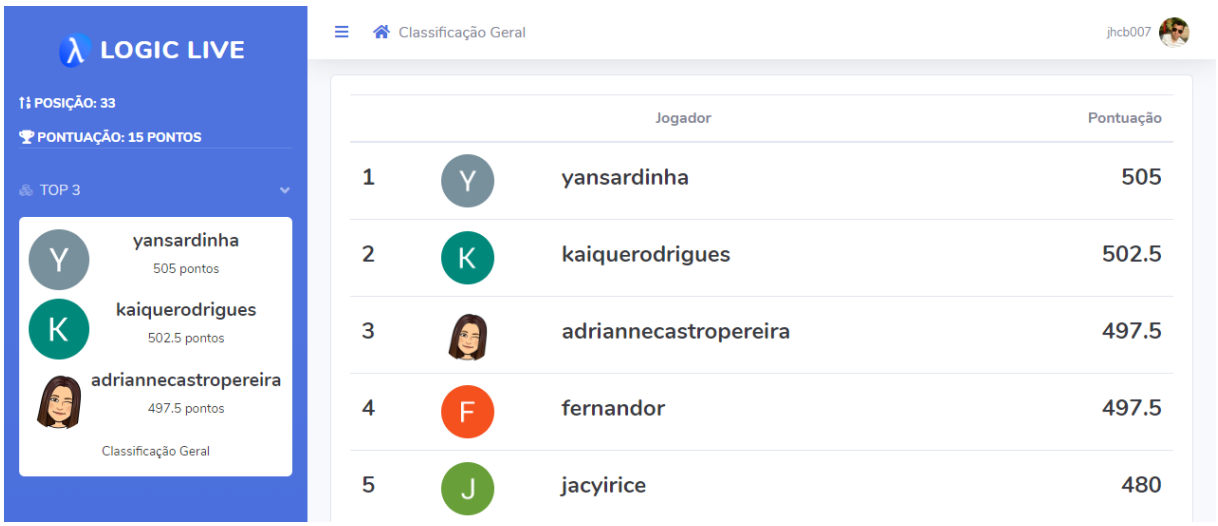


A Figura 17 apresenta a página inicial da plataforma, sem games cadastrados. Na lateral esquerda são exibidas informações sobre a posição atual do jogador na classificação geral, o total de pontuação e uma lista com os 3 jogadores que mais pontuaram na plataforma,

além de um *link* “Classificação Geral” que redireciona para página de classificação geral (Figura 18).

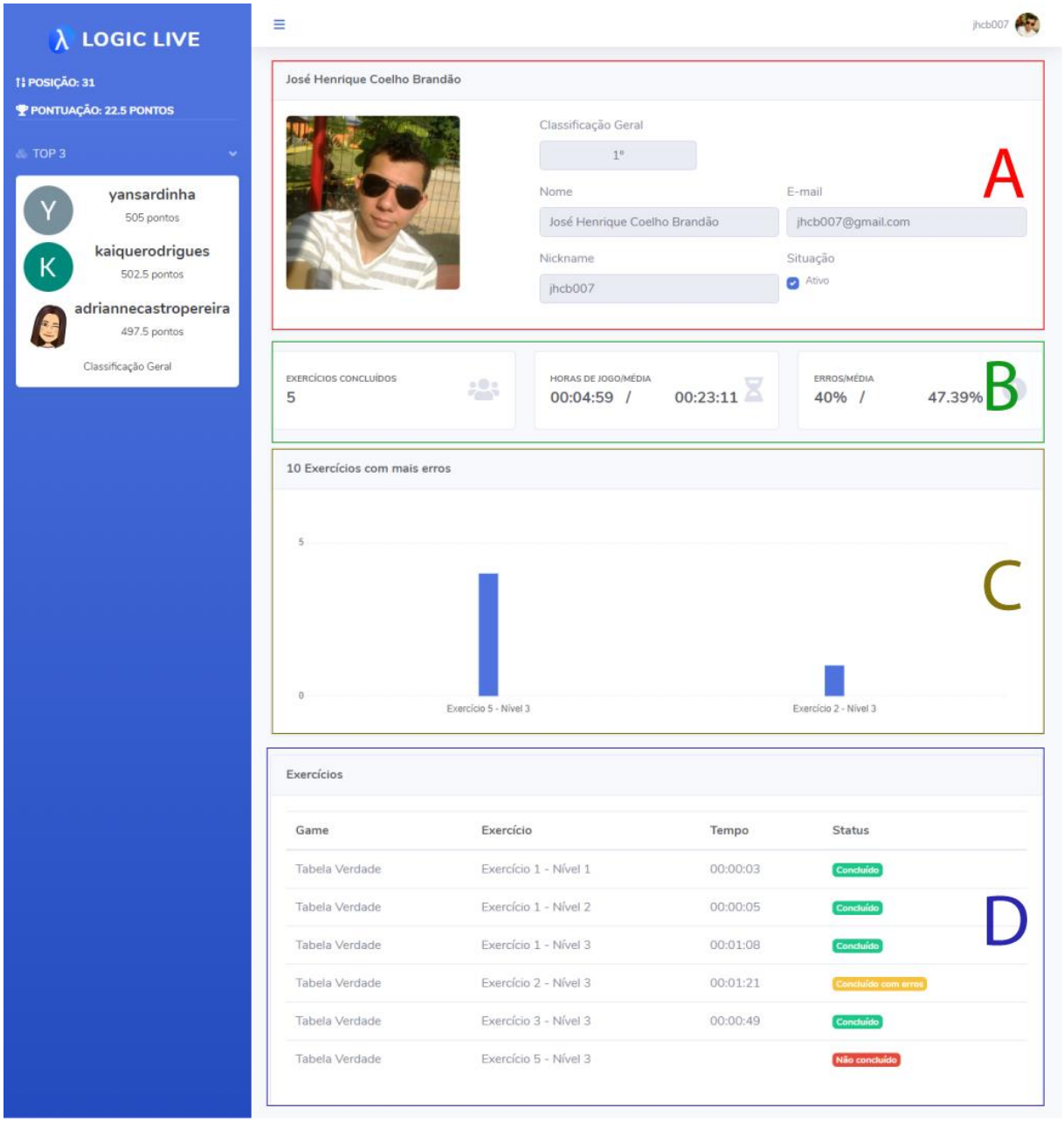
Na parte superior da página foi implementada a barra de informações, em que são apresentados dois botões, o primeiro corresponde ao botão de exibição do menu, e o segundo, que fica ao lado do nome de usuário que, ao ser clicado, exibirá os seguintes *links*: “Perfil”, redireciona o jogador para a página de perfil do jogador (Figura 19); “Sair”, finaliza a sessão do jogador na plataforma e o redireciona para a página de login (Figura 16).

Figura 18 - Página de classificação geral



Para que o jogador possa acompanhar a classificação de todos os jogadores da plataforma, a página de classificação geral foi implementada, como apresenta a Figura 18. Na página são exibidas as informações de posição, avatar, nome de usuário e pontuação geral de todos os jogadores ativos na plataforma.

Figura 19 - Página de perfil do jogador



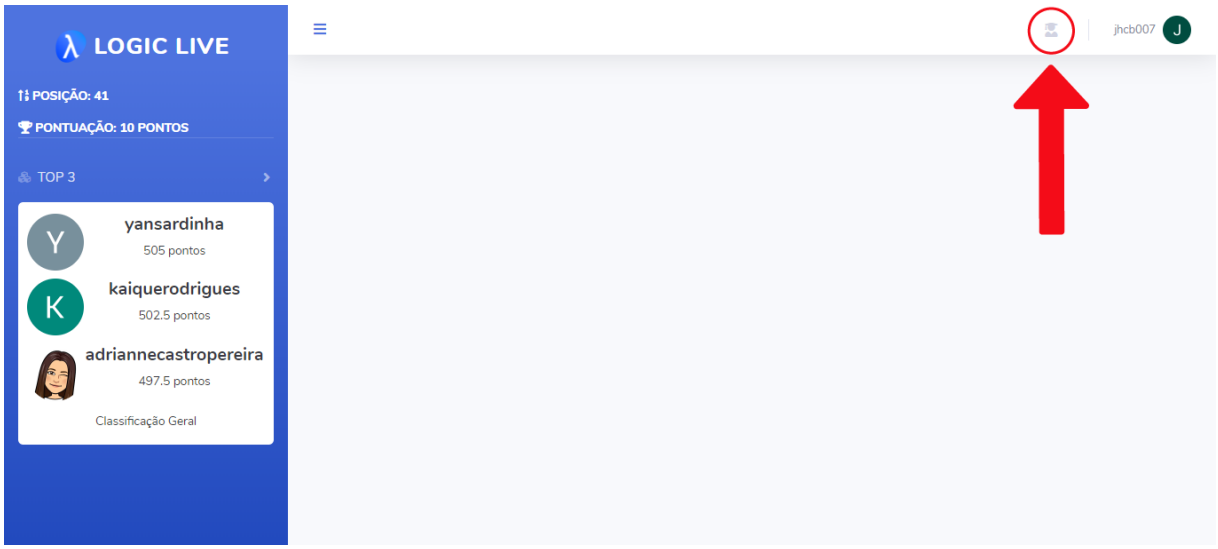
Com a finalidade de aumentar a motivação e o engajamento do jogador no processo de aprendizagem, além de realizar o *feedback* (retorno de informações relevantes ao jogador), o perfil do jogador, como apresenta a Figura 19, torna-se indispensável para aprimorar a experiência do jogador na plataforma.

Dividido em 4 grupos de *cards* (contêiner de conteúdo flexível e extensível), as informações estão dispostas na seguinte organização: 1º grupo (Figura 19-A), dados cadastrais do jogador, como avatar, classificação geral, nome, e-mail, nome de usuário e uma caixa de seleção na qual o jogador pode desativar ou ativar o seu perfil; 2º grupo (Figura 19-

B), conjunto de contadores com métricas referente ao quantitativo de exercícios concluídos, comparação entre as horas totais de jogo do jogador com a média geral de horas e comparação da porcentagem da assertividade dos exercícios com os demais jogadores; 3º grupo (Figura 19-C), apresenta o gráfico com os 10 exercícios que ocorreram mais erros; por fim o 4º grupo (Figura 19-D), exibe uma tabela com as informações referente aos exercícios jogados, exibindo o nome do game, exercício, tempo de conclusão do exercício e status (concluído, concluído com erros, não concluído) dos exercícios que o jogador interagiu.

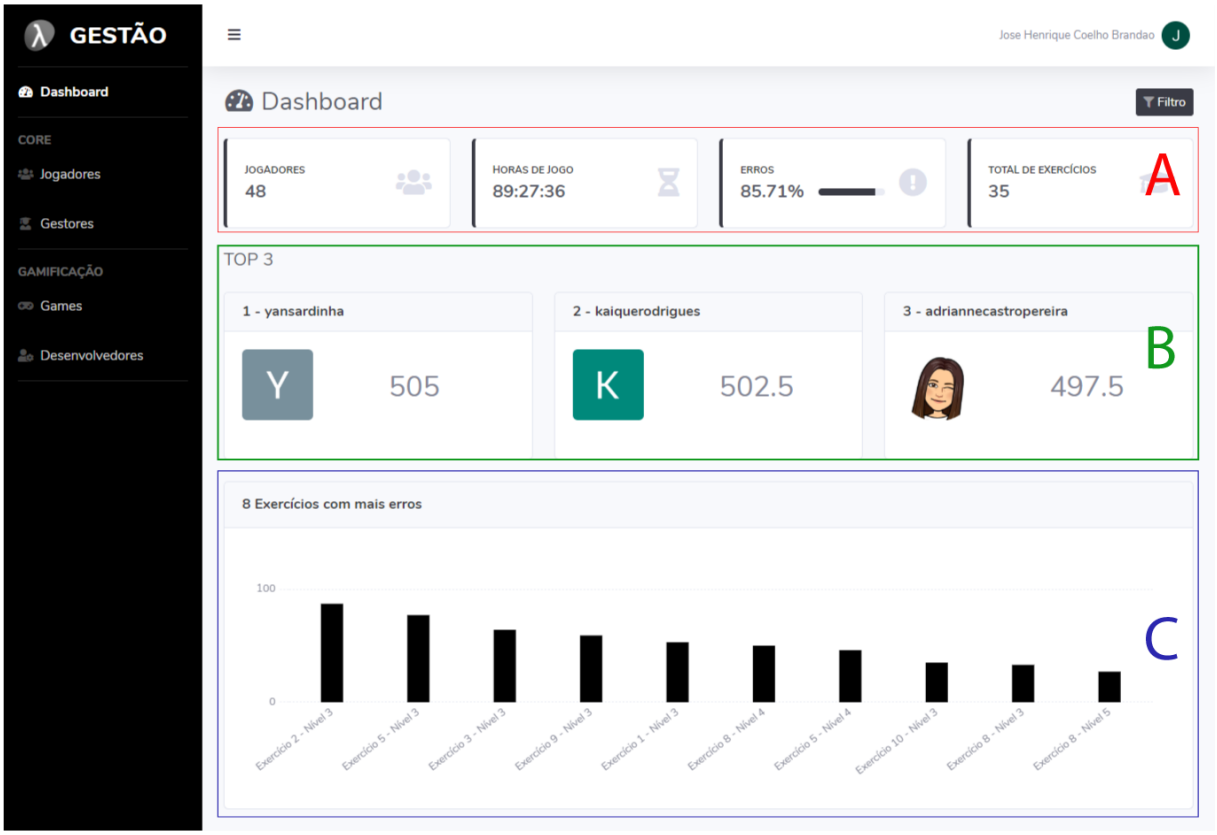
A Gestão da plataforma foi desenvolvida com o objetivo de monitorar o desempenho dos jogadores e visualizar os dados relativo ao processo de gamificação, permitindo o *feedback* em tempo real para os gestores/educadores.

Figura 20 - Acesso a página de Gestão



Para que os gestores possam acessar a parte de gestão da plataforma de forma mais fácil, foi estruturado que o acesso seguirá o mesmo procedimento apresentado na Figura 16, ao acessar um ícone será exibido na barra de informações com a descrição “Gestão”, como apresentado na Figura 20. Ao ser clicado o gestor é redirecionado para a página de *dashboard* na gestão (Figura 21).

Figura 21 - Página de dashboard

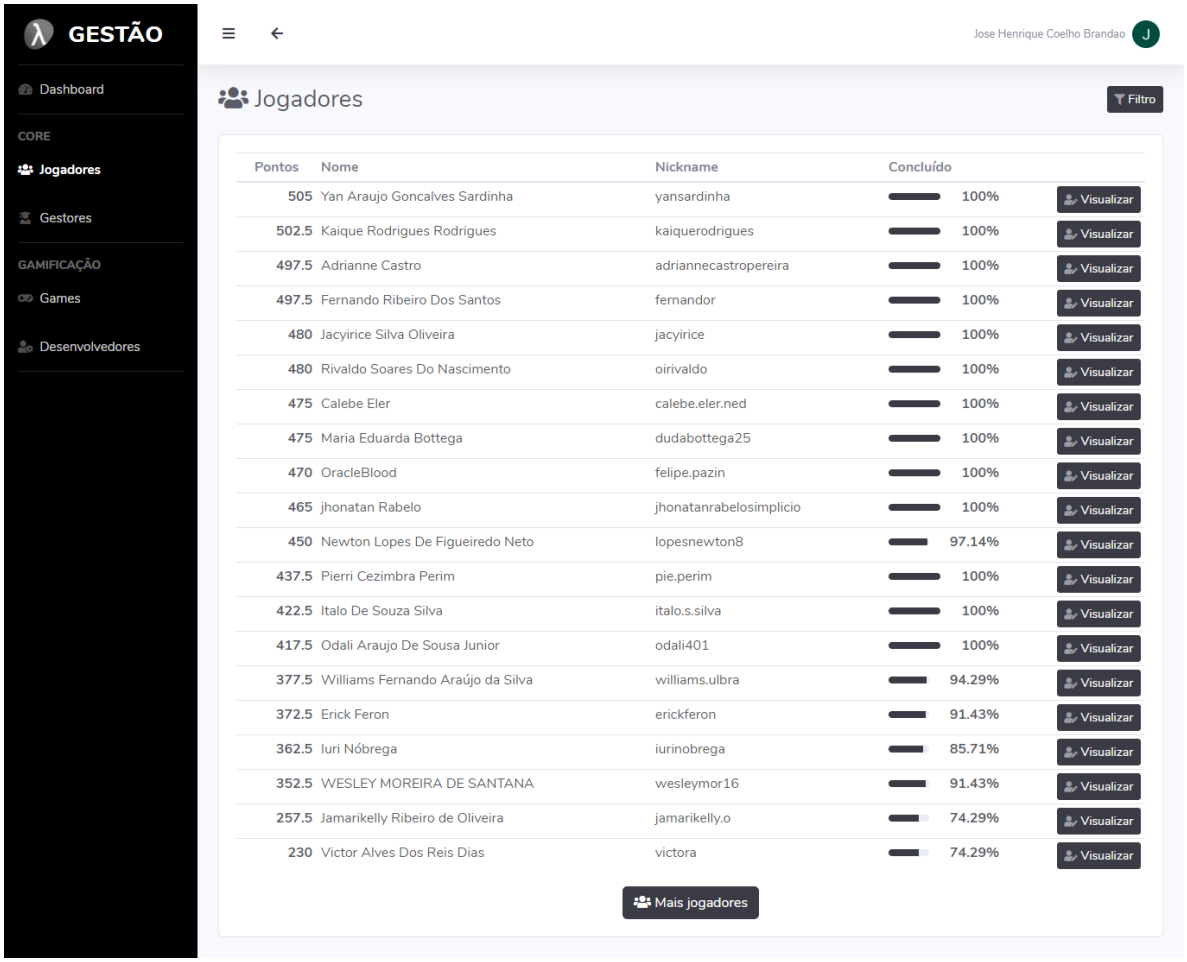


Ao acessar a Gestão, a página de *dashboard* é exibida, como apresentado na Figura 21**.** Na lateral esquerda é apresentado o menu dividido em dois grupos: core e gamificação. O primeiro é composto pelos *links*: “Jogadores”, redireciona o gestor para a página de listagem dos jogadores (Figura 22); “Gestores”, remete a página de listagem dos gestores (Figura 24). O segundo grupo trata da gestão do processo de gamificação, contido pelos *links*: “Games”, redireciona para a página de gerenciamento dos games (Figura 26); “Desenvolvedores”, encaminha a página de gerenciamento dos desenvolvedores (Figura 27).

Na parte central da página, um *dashboard* (painel visual) apresenta, de maneira centralizada, um conjunto de indicadores e métricas sobre o processo de gamificação e desempenho dos jogadores. Sua organização é composta por 3 grupos de *cards:* contadores (Figura 21-A), com métricas referente ao total de jogadores ativos na plataforma, horas totais de jogo de todos os jogadores, porcentagem de exercícios nos quais os jogadores cometeram pelo menos um erro e total de exercícios disponíveis na plataforma; classificatórios (Figura 21-B), apresentando em ordem decrescente os 3 jogadores com maior pontuação na

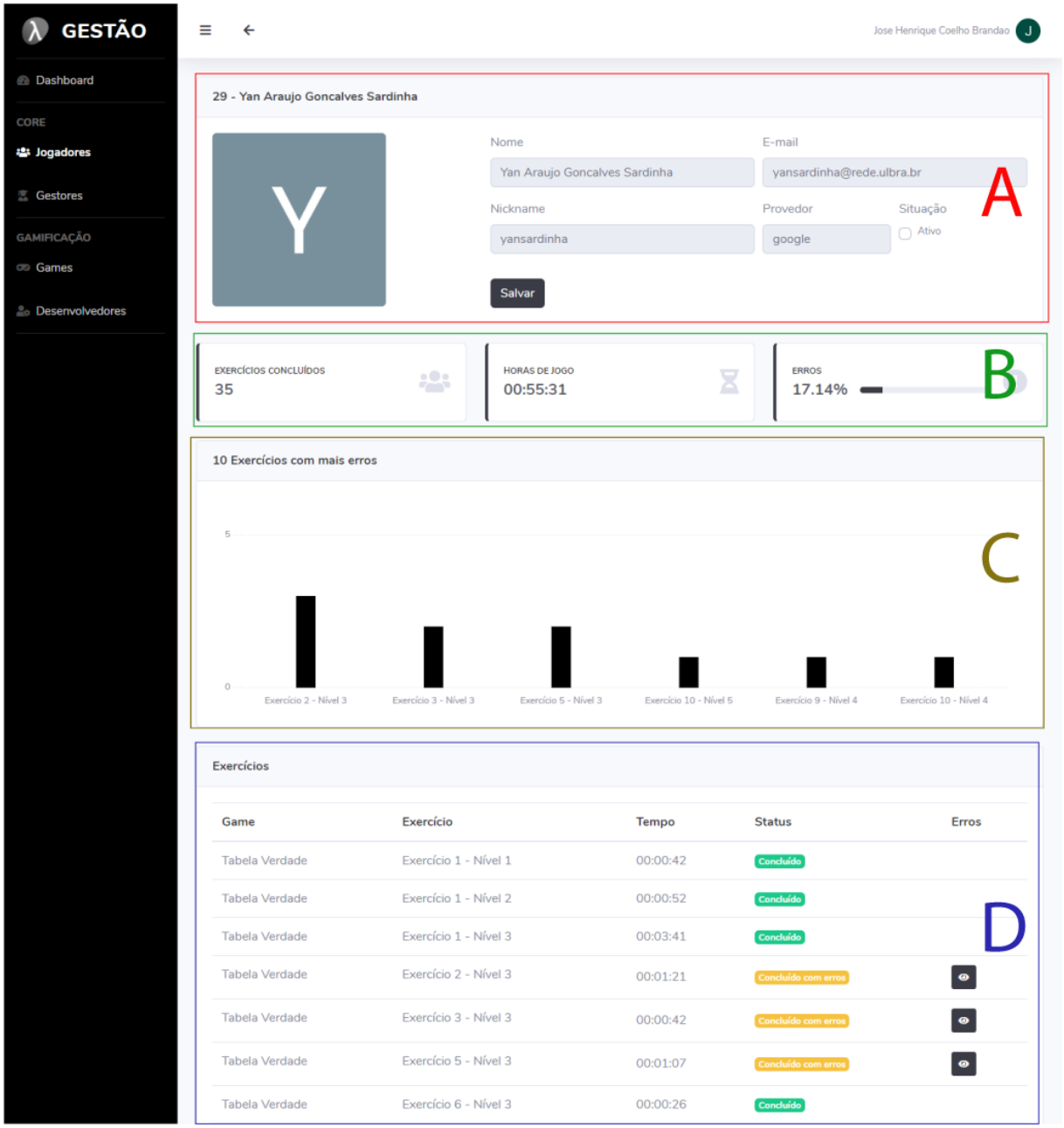
plataforma; por último o gráfico que exibe os 8 exercícios que ocorreram mais erros entre todos os jogadores (Figura 21-C).

Figura 22 - Página de listagem dos jogadores



A Figura 22 apresenta a listagem de todos os jogadores da plataforma em ordem decrescente de pontuação. A lista exibe o total de pontos, nome, nome de usuário, percentual de exercícios concluídos e um botão “Visualizar” que redireciona para página de perfil do jogador (Figura 23).

Figura 23 - Página de perfil do jogador em Gestão

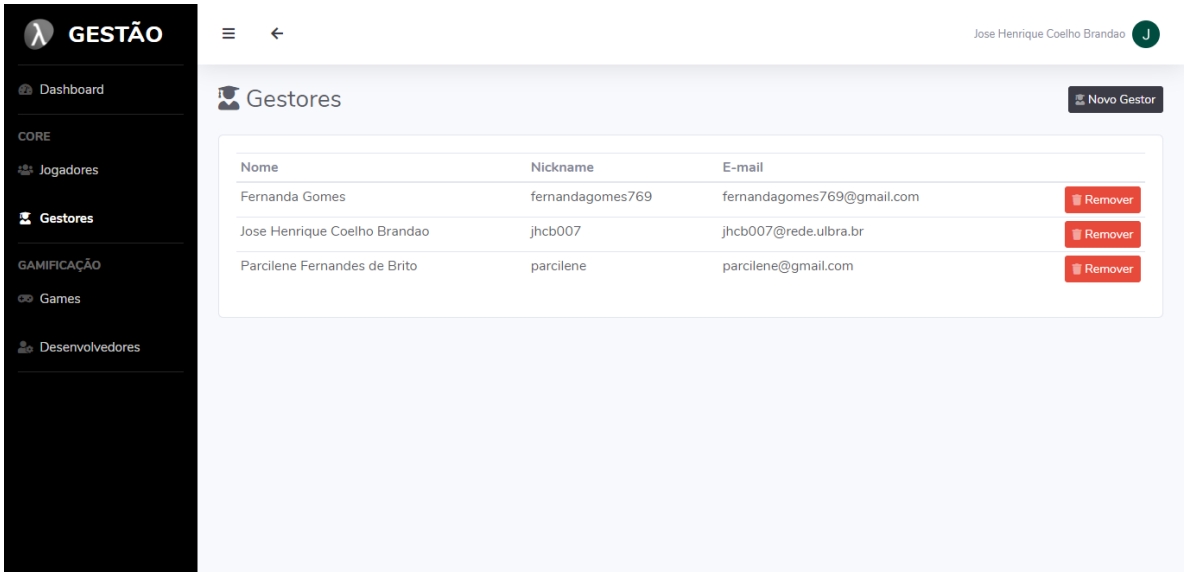


Para que os gestores/educadores possam acompanhar o engajamento e desempenho individual do jogador, a página de perfil do jogador foi desenvolvida, como apresenta a Figura 23. A página está organizada em 4 grupos de *cards* que dispõem sobre as informações do jogador e seu desempenho.

O primeiro grupo (Figura 23-A) ficou responsável por apresentar os dados cadastrais do jogador e a possibilidade de o gestor desativar ou ativar o perfil do jogador. Já o segundo grupo (Figura 23-B), exibe contadores com métricas sobre a quantidade de exercícios concluídos, total de horas jogadas e a porcentagem de exercícios que obtiveram respostas

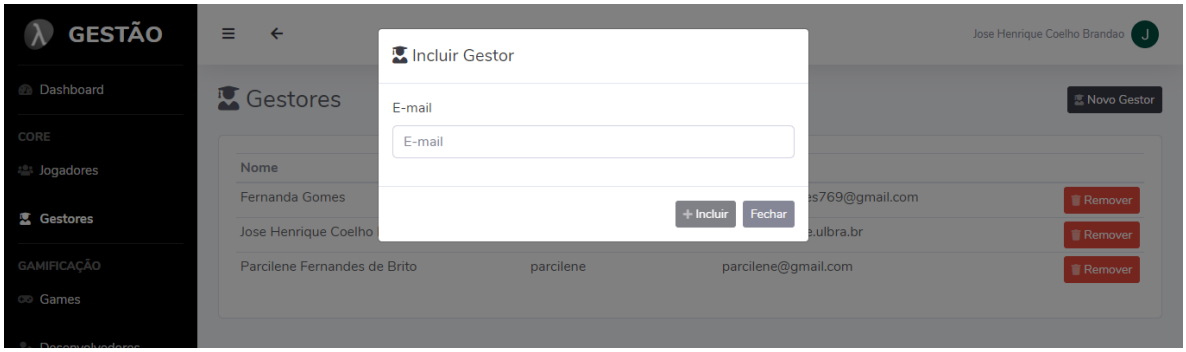
erradas. Em seguida, o terceiro grupo (Figura 23-C) apresenta o gráfico com os 10 exercícios que o jogador respondeu de forma errada. No último grupo de *cards* (Figura 23-D), a tabela de exercícios concluídos ou parcialmente concluídos é apresentada. Nela são exibidas as informações sobre o nome do game, exercício, tempo de jogo no exercício, status dos exercícios e o botão de "Erros" que possibilita ao gestor visualizar quais os erros cometidos pelo jogador no exercício.

Figura 24 - Página de listagem dos Gestores



A Figura 24 apresenta o botão "Novo Gestor", que possibilita a um jogador ter acesso a gestão (Figura 25). Além de exibir a listagem de todos os gestores cadastrado na plataforma em ordem alfabética. Na lista é possível visualizar o nome, nome de usuário, e-mail e o botão "Remover", que possibilita remover o acesso do gestor.

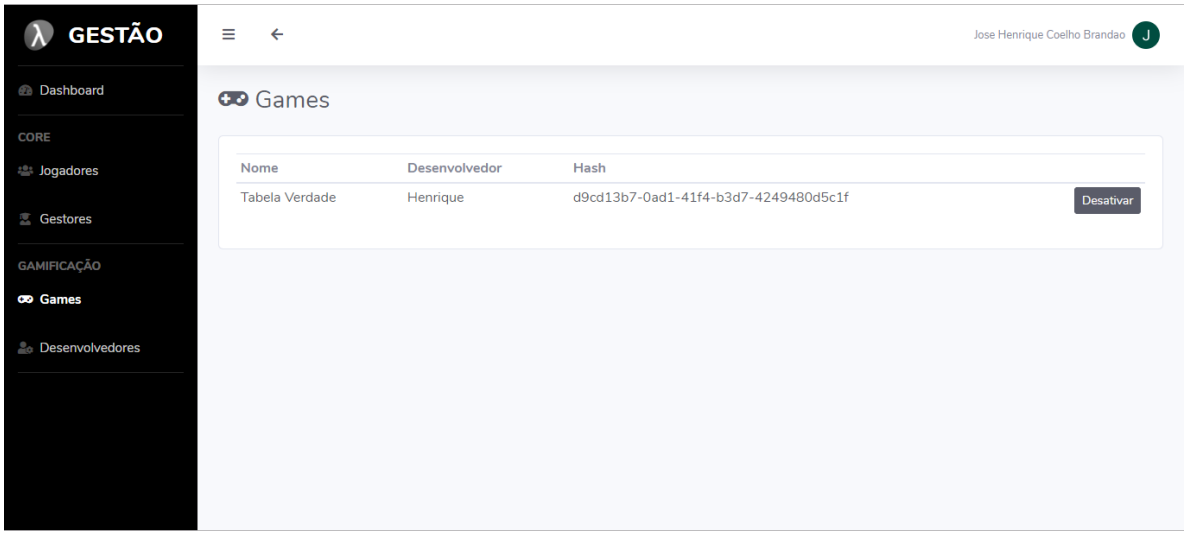
Figura 25 - Inclusão do Gestor



Para que o jogador possa ter acesso às páginas de gestão, é necessário informar o e-mail cadastrado na página de acesso da plataforma (Figura 16). Após informar o e-mail e

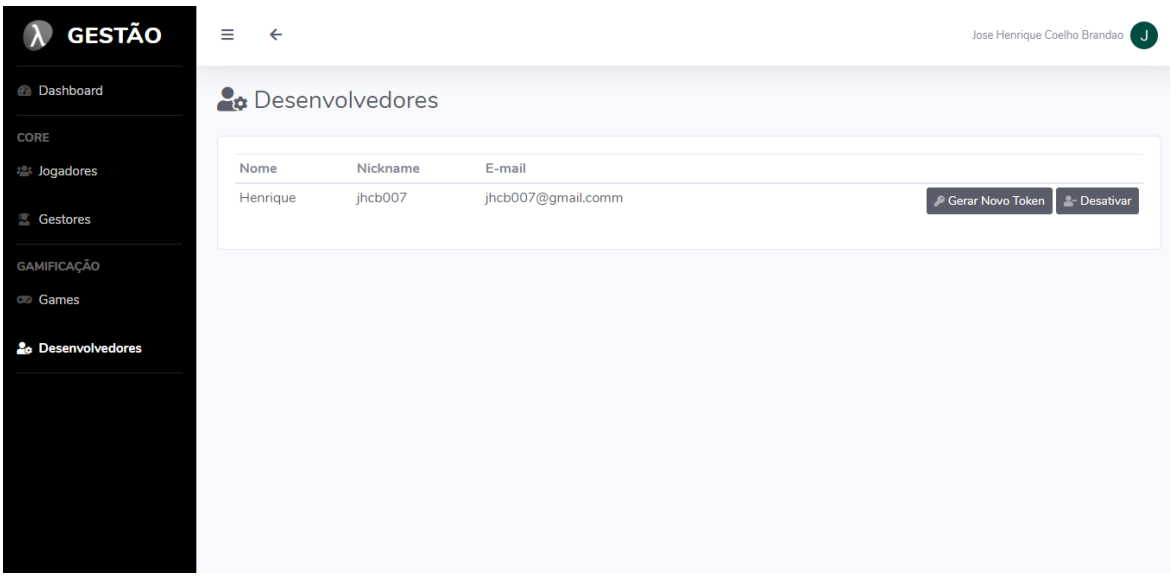
clicar no botão “Incluir”, o jogador passar a ser um gestor, podendo visualizar o desempenho dos demais jogadores e gerenciar a plataforma.

Figura 26 - Página de gerenciamento dos games



A Figura 26 exibe a listagem dos *games*, que são os módulos cadastrados pelos desenvolvedores, seguindo a documentação da plataforma (Figura 13). A listagem apresenta o nome do módulo, nome do desenvolvedor, *hash* do módulo e o botão “Desativar”, que possibilita ao gestor desativar o módulo na plataforma.

Figura 27 - Página de gerenciamento dos desenvolvedores



A Figura 27 mostra a página de gerenciamento dos desenvolvedores. A página exibe uma lista que apresenta as informações de nome, nome de usuário, e-mail, botão “Gerar Novo Token” e botão “Desativar” para cada desenvolvedor cadastrado. O primeiro botão executa

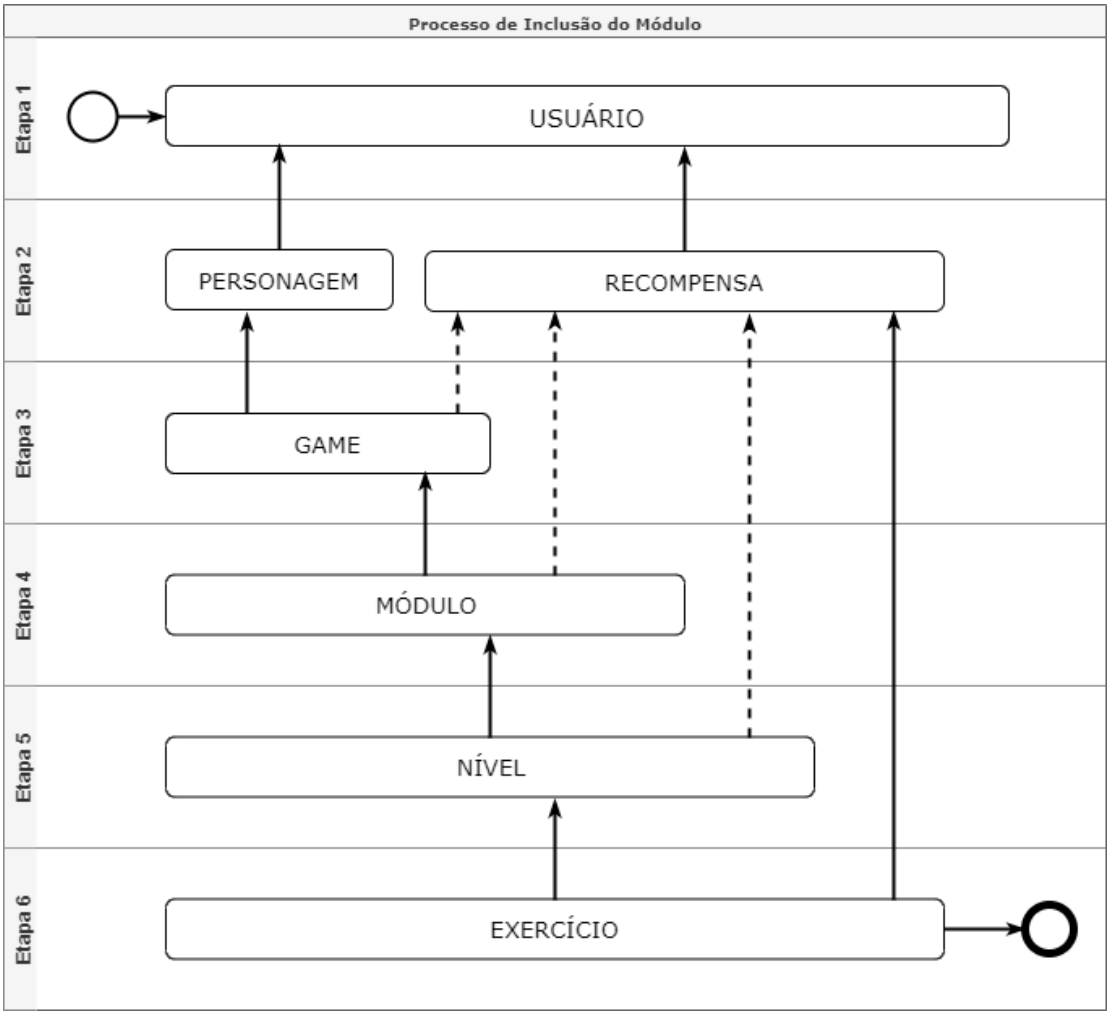
a ação de revogar os *tokens* ativos e gerar um novo *token* de acesso e o segundo fica responsável por desativar ou ativar o desenvolvedor na plataforma.

4.4 INSERÇÃO DO MÓDULO

Esta seção apresenta a inclusão na plataforma Logic Live o Módulo Tabela Verdade desenvolvido por Gomes (2019), bem como as alterações realizadas no *layout* do módulo para uma melhor experiência do jogador e funcionalidades do módulo já integrado a plataforma.

Utilizando a documentação apresentada na subseção 4.2, e aplicando o processo indicado na Figura 28, é possível acoplar de forma facilidada um novo módulo a plataforma.

Figura 28 - Processo de Inclusão do Módulo



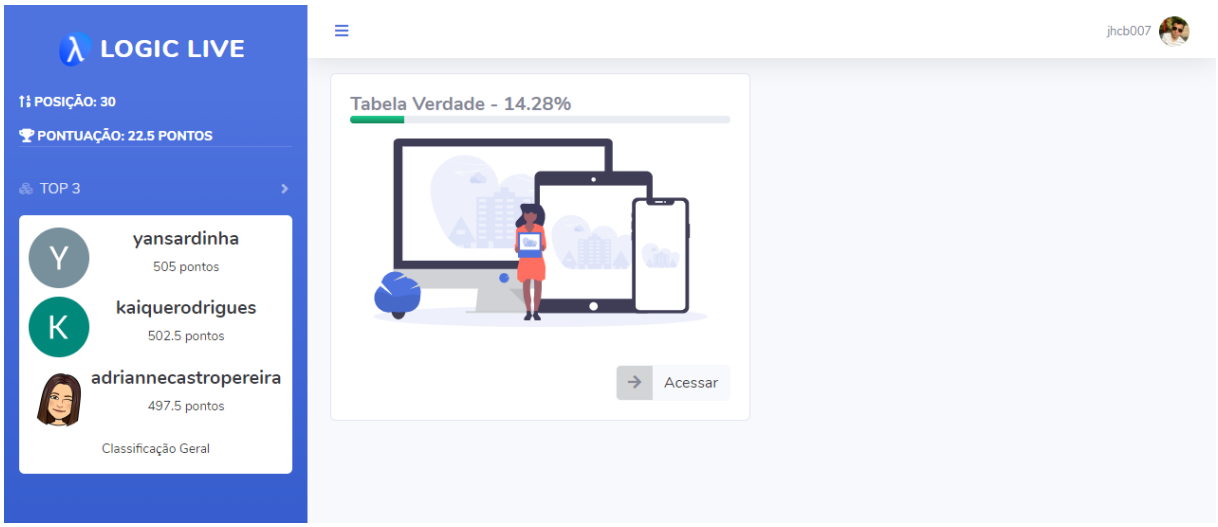
Para facilitar o entendimento do processo de inclusão de módulos, um esquema dividido em etapas foi desenvolvido, conforme mostra a Figura 28. As 6 etapas consistem em:

- etapa 1, início do processo, é onde o desenvolvedor cadastra os seus dados, como nome de usuário, e-mail e senha. Após realizar o cadastro, o desenvolvedor recebe um *token* que será utilizado nos próximos passos;
- etapa 2, cadastro do personagem ou recompensa, nesta fase não existe uma ordem de cadastro predeterminada, ficando a cargo do desenvolvedor escolher qual deseja cadastrar primeiro, a vinculação com o usuário é feita através do *token* de acesso, assim o desenvolvedor não precisa informar qual o código de usuário que cadastrou a recompensa ou o personagem;
- etapa 3, inclusão das informações do game, no caso do presente trabalho o game incluído foi o de Tabela Verdade (Figura 29), nesta fase foram informados o nome do game, uma breve descrição a respeito do game e o código do personagem que foi cadastrado na etapa anterior;
- etapa 4, inclusão do módulo, apesar do nome ser módulo, não são as informações a respeito do módulo Tabela Verdade, as informações já foram incluídas na etapa 3, os módulos são grupos de níveis, como no caso do game Tabela Verdade os grupos são: Estudo dos Conceitos da Lógica Proposicional, Criação de Tabelas e Identificação de Fórmulas e Verificação da Validade de Formas de Argumento;
- etapa 5, os níveis são basicamente agrupadores de exercícios, após cadastrar o módulo é necessário incluir os níveis, visto que o código do módulo é obrigatório no cadastro do nível;
- etapa 6, o último passo do processo consiste em incluir as informações do exercício, como código do nível a qual ele pertence, nome, descrição, tempo de execução máxima para resolver, código da recompensa e URL que será disponibilizado o exercício, por exemplo, no primeiro exercício do módulo Tabela Verdade, a URL utilizada foi *http://tabela-verdade.thelogiclive.com/#!/exercicios/basico/1*. Por motivo de conveniência, o exercício foi hospedado no mesmo servidor que a plataforma e um subdomínio *tabela-verdade* foi criado para distinguir a URL do módulo Tabela Verdade da plataforma Logic Live. A plataforma passa a ser totalmente desacoplada dos exercícios inseridos, tendo em vista que qualquer URL pode ser informada para um determinado exercício, logo o exercício pode ser hospedado em servidores diversos.

Vale ressaltar que a recompensa só é obrigatória no cadastro do exercício nos demais casos (game, módulo e nível) a sua vinculação é facultativa. Caso o desenvolvedor já tenha

feito o cadastro do seu usuário, a etapa 1 pode ser ignorada. Entretanto o módulo só estará disponível na plataforma, como é apresentado na Figura 29, se o processo for feito na sequência de etapas, conforme a Figura 28.

Figura 29 - Página inicial com módulo Tabela Verdade



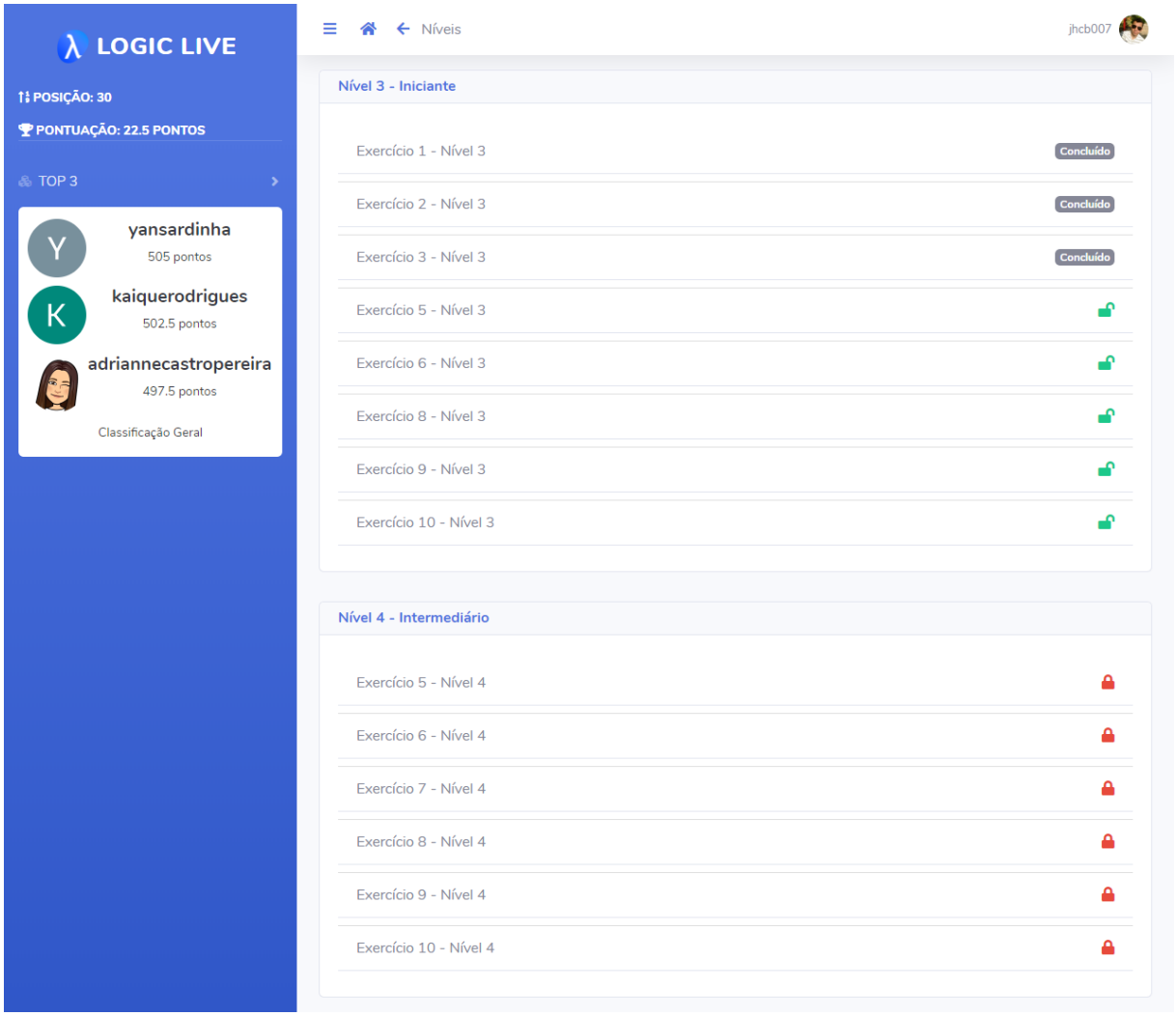
A Figura 29 apresenta a página inicial da plataforma com o Módulo de Tabela Verdade inserido. Nesta página, o componente *card* apresenta as informações sobre os módulos, são elas: barra de progresso e percentual de conclusão que indicará ao jogador o progresso atual no módulo, de forma que a barra e a porcentagem crescerá à medida que os exercícios do módulo são concluídos; imagem do módulo, para auxiliar o jogador a localizar de forma mais rápida qual módulo deseja jogar, levando em consideração que a plataforma pode receber inúmeros módulos e sua localização penas por nome pode ser prejudicada; botão “Acessar”, que redireciona o jogador para página de categorias dos exercícios (Figura 30).

Figura 30 - Página de categoria dos exercícios do módulo Tabela Verdade

A Figura 30 apresenta a página de categorias do Módulo de Tabela Verdade. As categorias foram divididas em *card*, possibilitando mais flexibilidade na exibição das informações, conforme modelo da figura anterior (Figura 29), os *cards* apresentam o percentual de exercícios concluídos nas categorias, a descrição das categorias e barra de progresso que reflete o percentual em forma gráfica.

Os *cards*, podem exercer dois estados: liberado, expõem as informações sobre o progresso do jogador na categoria e passa a assumir o comportamento de *links*, redirecionando para a lista de níveis e exercícios (Figura 31); bloqueado, as informações sobre o progresso são ocultadas e o ícone do cadeado é apresentado para facilitar a visualização do jogador. O estado de liberado só é assumido após a conclusão de todos os exercícios da categoria anterior.

Figura 31 - Página de listagem dos níveis e exercícios



A Figura 31 apresenta a página de listagem dos níveis e exercícios de uma determinada categoria (Figura 30) do Módulo Tabela Verdade. Nesta página, os níveis são agrupados por *cards* que contém a lista de exercícios. Cada item da lista exibe o nome do exercício e *status* (bloqueado, liberado, concluído). Quando o exercício assume o *status* de liberado, o item da lista passa o exercer o comportamento de *links*, direcionando o jogador para respectiva página do exercício (Figura 32).

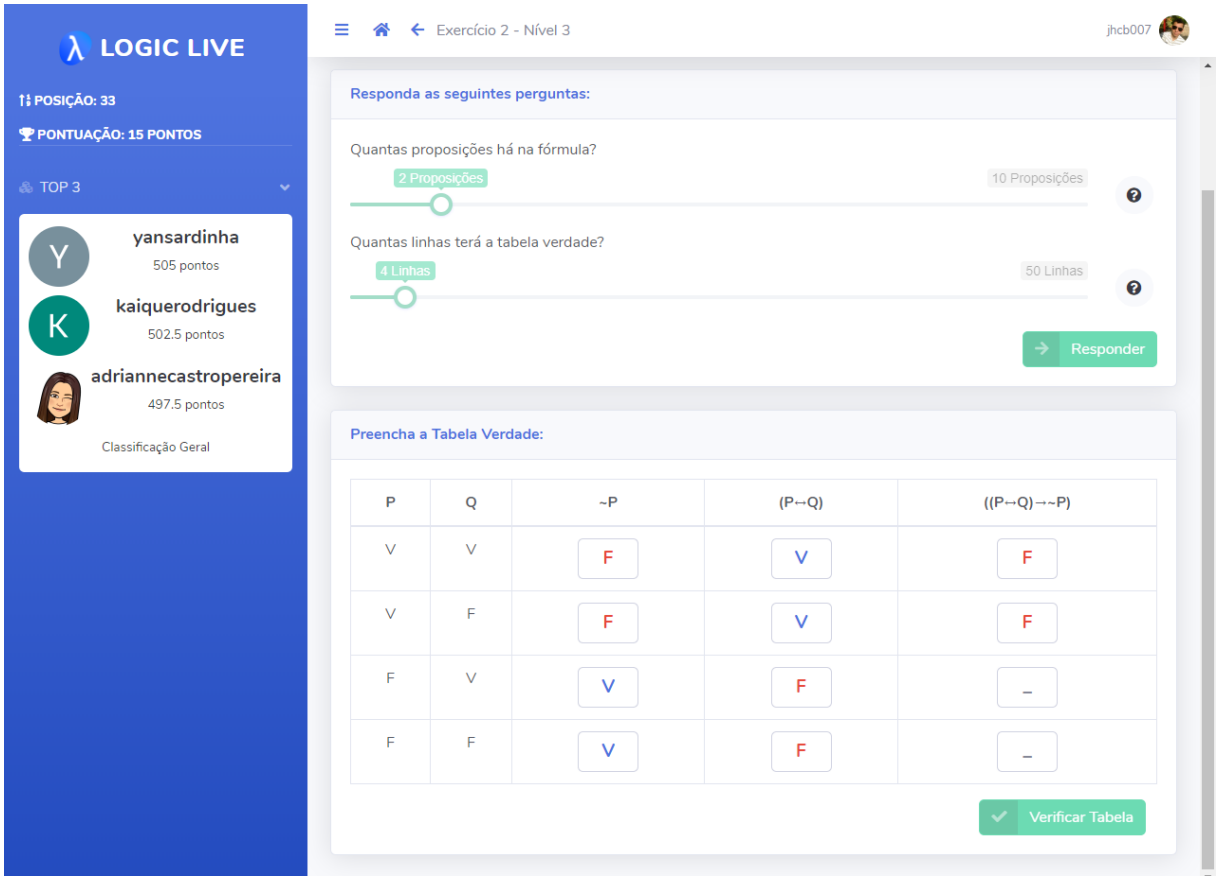
Na figura é possível observar que os três primeiros exercícios no Nível 3 foram concluídos e os demais apresentam o ícone de cadeado aberto na cor verde, que representa o *status* de liberado. Já no Nível 4, todos os exercícios estão bloqueados, exibindo o ícone do cadeado fechado na cor vermelha. Para que um nível passa a ter seus exercícios liberados, todos os exercícios do nível anterior, devem ser concluídos.

Figura 32 - Página do exercício do módulo Tabela Verdade

Como mostra a Figura 32, a página padrão de exercícios do Módulo Tabela Verdade, foi dividida em duas etapas. Na primeira etapa são apresentados dois *cards*, um para exibição da fórmula da Lógica Proposicional associada ao exercício e outro para que o jogador possa inserir nos campos indicados as respostas das perguntas com base na fórmula apresentada. O *layout* de inserção dos valores sofreu uma alteração para melhorar a experiência do usuário. No trabalho de Gomes (2019), os valores eram informados em um campo de texto, agora um controle deslizante (*range*) é apresentado, no qual o jogador deve deslizar até o valor desejado ser exibido. Se o jogador apresentar dificuldades para compreender as perguntas, é possível solicitar ajudar clicando no ícone de interrogação. Ao clicar no ícone uma caixa de diálogo com informações que auxiliem na resolução da pergunta são apresentadas.

A segunda etapa do exercício fica invisível até que o jogador responda corretamente às perguntas. Após as respostas serem verificadas como corretas, a segunda etapa passa a ser visível, como mostra a Figura 33.

Figura 33 - Página com a segunda etapa do exercício do módulo Tabela Verdade



A Figura 33 mostra a segunda etapa do exercício, apresentando a Tabela Verdade da fórmula. Contudo, somente os valores-verdade das proposições são exibidos, “V” para verdadeiro e “F” para falso, de forma que o jogador possa preencha a tabela atribuindo aos campos em branco os valores-verdade das subfórmulas. Os campos foram implementados de modo que apenas os valores “V” ou “F” serão aceitos. No trabalho de Gomes (2019), os valores não tinham cores atribuídas, o presente trabalho adicionou azul para valor “V” e vermelho para “F”, assim o jogador tem uma melhor visão da Tabela Verdade aumentando a usabilidade do módulo.

Após o jogador preencher todos os campos da Tabela Verdade, o botão “Verificar Tabela” é habilitado, ao clicar no botão, os valores informados serão verificados. Conforme a resposta o jogador, será apresentado uma notificação de “Resposta Incorreta” ou “Resposta Correta”, caso a resposta for correta, o jogador é direcionado para o próximo exercício, senão o jogador continua no exercício até respoder a tabela de forma correta.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o desenvolvimento de uma plataforma gamificada que permite de forma mais acessível a inclusão de novos módulos e elementos voltados ao processo de ensino e aprendizagem da disciplina de Lógica (Logic Live).

O uso da plataforma gamificada mostrou-se relevante pela facilidade em permitir analisar e visualizar métricas dos jogadores, pela integração de novos módulos em um único sistema, como também, a inclusão de novos objetivos e desafios, sem ter que partir para uma nova gamificação. Logo, implementar novos módulos no Logic Live passa a ser menos moroso, além do ganho de tempo no desenvolvimento.

Desenvolver a plataforma permitiu a mensuração do desempenho dos jogadores e a visualização dos dados relativo ao processo de gamificação que, em tempo real, possibilitou o *feedback* de cada perfil de jogador para os gestores/educadores. Além disso, foi possível oferecer recursos para integração e reutilização de componentes, empregando uma arquitetura baseada em serviço no desenvolvimento da plataforma, sendo possível agregar os módulos do Logic Live em um único sistema, reutilizar componentes dos módulos e disponibilizar uma API REST para analisar as métricas de desempenho dos jogadores.

A plataforma possibilita, também, manter os objetivos do jogo sempre alinhados com o interesse dos gestores/educadores, viabilizando análises, atualizações e criação de novos módulos sempre que necessário. Isso porque a utilização de serviços para analisar o desempenho dos usuários nos diversos módulos da plataforma e a reutilização de seus componentes facilitam o desenvolvimento mais dinâmico e rápido dos módulos. Por exemplo, a autenticação dos usuários é um serviço disponível na plataforma e pode ser utilizado por todos os módulos.

Na inclusão do módulo Tabela Verdade, foi verificado que permitir a utilização de diversos tipos de *layout* na construção de componentes impossibilitou o desenvolvimento de uma interface mais amigável entre o módulo e a plataforma. Um dos trabalhos futuros a serem realizado visa desenvolver um estudo para verificar qual padrão de *layout*, como cores e tamanho de componentes a ser utilizados pelos módulos que serão incluídos na plataforma, objetivando o aprimoramento da interface do jogador e, assim, melhorar a experiência de jogabilidade que eleva o engajamento do jogador.

Outro trabalho futuro a ser realizado está relacionado à utilização da plataforma por meio de *smartphone*. Levando em consideração o grande aumento na utilização de *smartphone* no Brasil, principalmente por jovens. Nesse sentido, a partir de um estudo sobre técnicas de desenvolvimento para aplicativos *mobile*, poderia ser disponibilizado um padrão de *design* para os módulos que seja acessível por *smartphone.* Desta forma, um maior número de jogadores poderiam ser atingidos, contribuir para o aumento de ensino-aprendizagem das disciplina de Lógica (Logic Live).

REFERÊNCIAS

ADONISJS. **About AdonisJs.** Disponível em: <https://adonisjs.com/docs/4.1/about>. Acesso em: 14 abr. 2019.

AL-KOFAHI, Majd Mahmoud. **Service Oriented Architecture (SOA) Security Models.** 2011. 128 f. Dissertação (Mestrado) - Curso de Computer Engineering, Iowa State University, Ames, Iowa, 2011. Disponível em: <https://lib.dr.iastate.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=3006&context=etd>. Acesso em: 10 maio 2019.

ALLEN, Robert; GARLAN, David. A formal basis for architectural connection. **Acm Transactions On Software Engineering And Methodology**, [s.l.], v. 6, n. 3, p.213-249, 1 jul. 1997. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/258077.258078.

AZARITE, Ricardo. **Gamification: como gerar engajamento com jogos nas redes sociais**. 2013. Disponível em: <http://ideas.scup.com/pt/files/downloads/2013/05/gamification_-scupi deas.pdf>. Acesso em: 15 set. 2018.

BASS, Len; KAZMAN, Rick; CLEMENTS, Paul. **Software Architecture In Practice.** Pittsburgh: Addison-wesley Longman, 2002.

COSTA, Amanda Cristina Santos; MARCHIORI, Patrícia Zeni. Gamificação, elementos de jogos e estratégia: uma matriz de referência. **Incid**: Revista de Ciência da Informação e Documentação, [s.l.], v. 6, n. 2, p.44-65, 2 out. 2015. Universidade de São Paulo Sistema Integrado de Bibliotecas - SIBiUSP. http://dx.doi.org/10.11606/issn.2178-2075.v6i2p44-65. Disponível em: <http://www.revistas.usp.br/incid/article/view/89912>. Acesso em: 25 set. 2018.

CUNHA, Lucas Felipe da; GASPARINI, Isabela; BERKENBROCK, Carla D. M. Investigando o uso de gamificação para aumentar o engajamento em sistemas colaborativos.

Proceedings Of The V Workshop Sobre Aspectos da Interação Humano-computador na Web Social, Manaus, p.28-33, 2013. Disponível em: <http://ceur-ws.org/Vol-1051/paper4.pdf>. Acesso em: 21 mar. 2019.

DEGAN, Joyce Otsuka Cortes. **Integração de dados corporativos: uma proposta de arquitetura baseada em serviços de dados.** 2015. 118 f. Dissertação (Mestrado) - Curso de Engenharia da Computação, Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2015. Disponível em: <http://repositorio.unicamp.br/handle/REPOSIP/276489>. Acesso em: 14 abr. 2019.

DETERDING, Sebastian et al. Gamification. using game-design elements in non-gaming contexts. **Proceedings Of The 2011 Annual Conference Extended Abstracts On Human Factors In Computing Systems - Chi Ea '11**, [s.l.], p.2425-2428, maio 2011. ACM Press. http://dx.doi.org/10.1145/1979742.1979575.

DOCS, Mdn Web. **Uma visão geral do HTTP.** 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 09 maio 2019.

FIELDING, Roy Thomas. **Architetural Styles and the Design of Network-based Software Architectures.** 2000. Dissertação (Doutorado em Filosofia da Computação). Universidade da Califórnia. Irvine.

GAMMA, Erich et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Holland: Addison-wesley Professional, 1994.

GHEZZI, Carlo; JAZAYERI, Mehdi; MANDRIOLI, Dino. **Fundamentals of Software Engineering.** Michigan: Prentice Hall, 1991. 573 p.

GOMES, Fernanda Pereira. **Desenvolvimento de um Framework para gamificação baseada na Tríplice Contingência e aplicação no Módulo de Tabela Verdade do Logic Live**. 2019. 80 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas/TO, 2019.

HAGGLUND, Per. **Taking gamification to the next level**: A detailed overview of the past, the present and a possible future of gamification. 2012. 36 f. Umeå Universitet, Suécia, 2012. Disponível em: <http://www.diva-portal.org/smash/get/diva2:5 46713/FULLTEXT01.pdf>. Acesso em: 22 set. 2018.

HANSEN, Roseli P; Pinto, Sérgio Crespo S. C. (2003) "**Construindo Ambientes de Educação Baseada na Web Através de Web Services Educacionais**", XIV Simpósio Brasileiro de Informática na Educação, RJ, novembro de 2003.

HERZIG, Philipp; AMELING, Michael; SCHILL, Alexander. A Generic Platform for Enterprise Gamification. **2012 Joint Working Ieee/ifip Conference On Software Architecture And European Conference On Software Architecture**, [s.l.], p.219-223, ago. 2012. IEEE. http://dx.doi.org/10.1109/wicsa-ecsa.212.33.

HUHNS, M.N.; SINGH, M.P.. Service-oriented computing: key concepts and principles. **Ieee Internet Computing**, [s.l.], v. 9, n. 1, p.75-81, jan. 2005. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mic.2005.21.

KREGER, Heather. **Web Services Conceptual Architecture (WSCA 1.0)**. IBM Software Group, Somers, Ny, v. 1, n. 5, p.6-40, maio 2011.

LIMA, Lucas Ferreira de. **CONTRATOS REST ROBUSTOS E LEVES:** UMA ABORDAGEM EM DESIGN-BY-CONTRACT COM NEOIDL. 2016. 101 f. Dissertação (Mestrado) - Curso de Faculdade de Tecnologia, Departamento de Engenharia ElÉtrica, Universidade de Brasília, Brasília, 2016. Disponível em: <http://repositorio.unb.br/handle/10482/22101>. Acesso em: 13 set. 2019.

NASCIMENTO, Wellington. **Documentação de APIs com Swagger no ASP.Net Core.** 2018. Disponível em: <https://www.wellingtonjhn.com/posts/documenta%C3%A7%C3%A3o-de-apis-com-swagger-no-asp.net-core/>. Acesso em: 13 set. 2019.

MATOS, Jonathan de. **Distribuição de carga flexível e dinâmica para provedores de web services**. 2009. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2009. doi:10.11606/D.55.2009.tde-07052009-100457. Acesso em: 2019-05-21.

MENEZES, Graciela Sardo et al. Reforço e Recompensa: a Gamificação tratada sob uma abordagem behaviorista. **Projetica**. Londrina, p. 09-18. 18 dez. 2014. Disponível em: <http://www.uel.br/revistas/uel/index.php/projetica/article/viewFile/17746/16089>. Acesso em: 16 set. 2018.

MILANI, André. **MySQL:** Guia do programador. São Paulo: Novatec, 2006.

MONROE, R.T. et al. Architectural styles, design patterns, and objects. **Ieee Software**, [s.l.], v. 14, n. 1, p.43-52, 1997. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/52.566427.

NITTO, Elisabetta Di; ROSENBLUM, David. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. **Proceedings Of The 21st International Conference On Software Engineering - Icse '99**, Los Angeles, v. 1, n. 21, p.13-22, maio 1999. ACM Press. http://dx.doi.org/10.1145/302405.302406.

NODEBR (Org.). **O que é Node.js?** Disponível em: <http://nodebr.com/o-que-e-node-js/>. Acesso em: 14 abr. 2019.

PAULA, Bruno Henrique de; VALENTE, José Armando; BURN, Andrew. Game-making as a means to deliver the new Computing Curriculum in England. **Currículo Sem Fronteiras**, v. 14, n. 3, p.46-69, set. 2014. Disponível em: <http://www.curriculosemfronteiras.org/ vol14iss3articles/paula-valente-burn-en.pdf>. Acesso em: 18 mar. 2019.

PERRY, Dewayne E.; WOLF, Alexander L. Foundations for the study of software architecture. **Acm Sigsoft Software Engineering Notes**, [s.l.], v. 17, n. 4, p.40-52, 1 out. 1992. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/141874.141884.

PRO, Ludos. **Plataforma de Gamificação: Entenda o que é e como funciona**. 2019. Disponível em: <https://www.ludospro.com.br/blog/plataforma-de-gamificacao>. Acesso em: 18 mar. 2019.

SESHADRI, Shyam; GREEN, Brad. **Desenvolvendo com AngularJS.** São Paulo: Novatec, 2014.

SILVA, Alan de Oliveira. **Design by Contract em Web Services RESTful.** 2017. 63 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2017.

SILVA, Ricardo Frenedoso da; GONÇALVES, Pablo Rodrigo. Web Services: Uma Análise Comparativa. **Revista das Faculdades Integradas Claretianas**, São Paulo, Sp, n. 5, p.7-19, dez. 2012. Disponível em: <https://www.researchgate.net/profile/Pablo_Goncalves/publication/273316362_Web_Services_-_Uma_Analise_Comparativa/links/54fdb0990cf270426d12c969.pdf>. Acesso em: 15 maio 2019.

SHAW, M. Comparing architectural design styles. **Ieee Software**, [s.l.], v. 12, n. 6, p.27-41, 1995. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/52.469758.

SRIPADA, Sai Krishna; REDDY, Y. Raghu; KHANDELWAL, Shivam. Architecting an extensible framework for Gamifying Software Engineering concepts. **Proceedings Of The 9th India Software Engineering Conference On - Isec '16**, [s.l.], p.119-130, 2016. ACM Press. http://dx.doi.org/10.1145/2856636.2856649.

STAL, Michael. Web services: beyond component-based computing. **Communications Of The Acm**, [s.l.], v. 45, n. 10, p.71-76, 1 out. 2002. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/570907.570934.

STEFANONI, Giuseppe; STAGLIANÒ, Loris. **Designing enterprise gamification architectures**. 2014. 178 f. Tese (Doutorado) - Curso de Engenharia da Informação, Politecnico di Milano, Milão, 2013. Disponível em: <https://www.politesi.polimi.it/handle/10589/86122?mode=simple&submit_simple=Visualizza+la+scheda+semplice+del+documento>. Acesso em: 18 mar. 2019.

STREIBEL, M. **Implementando Web Services. Trabalho de Especialização,** Universidade. Federal do Rio Grande do Sul, Porto Alegre, 2005.

SYSTINET. **Introduction to Web Services Architecture.** Cambridge, Massachusetts: Systinet Corp, 2002. Disponível em: <https://immagic.com/eLibrary/ARCHIVES/GENERAL/SYSTINET/S020716S.pdf>. Acesso em: 15 maio 2019.

TODA, A., SILVA, Y.R., CRUZ, W., XAVIER, L., ISOTANI, S. Um processo de Gamificação para o ensino superior: Experiências em um módulo de Bioquímica. IN: **Anais do Workshop de Informática na Escola**, p. 495 (2016) Disponível em: <http://www.br-ie.org/pub/index.php/wie/article/view/6856/4734>. Acesso em: 12 abr. 2019.

THE INTERNET SOCIETY. **Hypertext Transfer Protocol:** HTTP/1.1. 1999. Disponível em: <https://www.w3.org/Protocols/rfc2616/rfc2616.html>. Acesso em: 09 maio 2019.

WERBACH, K.; HUNTER, D. **For the win**: how game thinking can revolutionize your business. Philadelphia: Wharton Digital Press, 2012.

VIANNA, Ysmar et al. Gamification, Inc.: **Como reinventar empresas a partir de jogos**. Rio de Janeiro: Mjv Press, 2013. 116 p.

WOOD, Lincoln C.; REINERS, Torsten. Gamification. **Encyclopedia Of Information Science And Technology, Third Edition**, [s.l.], p.3039-3047, jan. 2015. IGI Global. http://dx.doi.org/10.4018/978-1-4666-5888-2.ch297.

W3C. **SOAP Version 1.2 Part 1:** Messaging Framework (Second Edition). Disponível em: <https://www.w3.org/TR/soap12-part1/>. Acesso em: 14 maio 2019.

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

ZUR MUEHLEN, Michael; NICKERSON, Jeffrey V.; SWENSON, Keith D. Developing web services choreography standards—the case of REST vs. SOAP. **Decision Support Systems**, [s.l.], v. 40, n. 1, p.9-29, jul. 2005. Elsevier BV. http://dx.doi.org/10.1016/j.dss.2004.04.008.