CodeLive: Ambiente gamificado para o aprendizado de programação

Jesiel Souza Padilha[1], Jackson Gomes de Souza[2], Fabiano Fagundes[2], Madianita Bogo Marioti[2]

[1]Acadêmico do Curso de Ciência da Computação

[2]Professor (a) dos cursos de Ciência da Computação e Sistemas de Informação

jesielpadilha.ti@gmail.com,jgomes@ceulp.edu.br,{thilfa,madianitab}@gmail.com

Abstract. *Cardoso (2015) initiated and Padilha (2016) developed the CodeLive project, aiming to simplify computer programming teaching and learning. Cardoso (2015) used elements from gamification and space battle as a metaphor to conceive a web based software which main purpose is to motivate the student in computer programming learning and practice task in a playful way. This paper focuses on the development process and presents results obtained from updates on CodeLive. Such updates were applied in order to solve two problems: a) to provide better interaction among user and software; and b) to give the user a code execution tool able to evaluate the correctness of user's input.*

Resumo. *Buscando simplificar o modo como se aprende e se pratica programação, surgiu o projeto CodeLive, criado por Cardoso (2015) e desenvolvido por Padilha (2016). Utilizando os elementos da gamificação e batalha espacial como metáfora, Cardoso (2015) concebeu um software web que tem como proposta motivar o usuário a aprender e praticar programação de maneira lúdica. Tal trabalho trata do processo de desenvolvimento e dos resultados obtidos após uma atualização da ferramenta citada. Essa atualização buscou sanar dois problemas existentes: proporcionar ao usuário maior interação e fornecer ao mesmo uma ferramenta de execução de códigos eficiente e que fosse capaz de validar a resposta informada pelo usuário.*

1. Introdução

A ferramenta CodeLive é um ambiente online para aprendizagem e prática de linguagem de programação. Foi criada com base em uma ferramenta, chamada JLive 2, desenvolvida por professores dos cursos de Sistemas de Informação e Ciência da Computação da instituição de ensino de Padilha (2016), e utilizada como suporte pedagógico em disciplinas com foco no ensino de algoritmos e linguagem de programação. A ferramenta não possuía recursos que a tornasse atraente para o usuário, pois tinha um comportamento muito similar ao de IDEs de desenvolvimento tradicionais. Com foco nesse contexto Cardoso (2015) trabalhou para transformar um então ambiente virtual de programação JLive 2 em um ambiente gamificado e colaborativo.

Para que os objetivos de Cardoso (2015) fossem alcançados foram utilizados os elementos da gamificação e batalha espacial como metáfora, proporcionando a criação de um software web cuja a proposta é motivar o usuário a aprender e praticar programação de maneira lúdica. Tal objetivo foi alcançado por meio de uma mecânica de jogo que inclui elementos como: nível de experiência, domínio da força e recompensas, que o usuário ganha a cada desafio vencido. Os desafios são exercícios de programação, logo, resolvê-los aumenta o conhecimento do usuário e faz com que o mesmo avance dentro do jogo (por exemplo, subindo de nível).

O presente trabalho teve a tarefa de dar continuidade ao desenvolvimento do CodeLive com o intuito de torná-lo mais eficiente no que tange à interatividade com o usuário e à execução do código inserido pelo mesmo ao responder um desafio. O objetivo principal, foi utilizar tecnologias e ferramentas que permitissem diminuir o tempo de resposta, apresentar notificações de atividades de outros usuários e utilizar elementos mais comuns em softwares web modernos (como o conceito de tela responsiva, carregamento progressivo e comunicação assíncrona com o servidor ou *back-end*).

Para alcançar os objetivos foi preciso entender melhor a respeito de tecnologias de comunicação cliente-servidor na web. Como o protocolo HTTP é o mais comum nesse contexto, foi necessário estudar mais detalhadamente seus recursos.

O HTTP é um protocolo de comunicação entre cliente-servidor baseado no paradigma *request/response* (ou requisição-resposta), o mesmo é genérico, orientado a objetos e *stateless* (sem estado) [ROSS e KUROSE 2013]. Nele o cliente realiza uma requisição ao servidor e este por sua vez a responde, porém, toda a comunicação se mantém "parada" até que uma nova ação seja feita pelo cliente [MULLER 2014]. Logo o HTTP não permite que servidores iniciem a comunicação com os clientes e aplicações que exigem essa característica precisam de alternativas como o modelo de comunicação *push server*. O *push server* é uma técnica criada com base no modelo "comet", apresentada em 2006 por Alex Russell. O "comet" descreve um modelo de comunicação entre cliente e servidor em que a cada requisição do cliente a conexão HTTP é persistida por um certo tempo, permitindo que nesse período o servidor envie dados para o cliente sem uma requisição HTTP explícita [ALVARENGA 2013].

Diante do conhecimento acerca das limitações do protocolo HTTP para o desenvolvimento de software web interativo e da carência de uma melhora na interação da ferramenta com o usuário, entendeu-se que este ambiente poderia proporcionar uma comunicação com o usuário que fosse além da tradicional. Por exemplo, anteriormente o usuário precisava acessar a "página de desafios" para saber quais novos itens estavam disponíveis, porém seria mais interessante uma funcionalidade em que o próprio CodeLive notificasse o usuário quando um novo desafio estivesse disponível, sem a necessidade de uma ação do usuário para tomar conhecimento deste fato.

Para solucionar a questão da melhoria da interatividade com o usuário foi utilizada a tecnologia *WebSocket*. Para Alvarenga (2013), *WebSocket* é uma tecnologia que permite a comunicação bidirecional entre clientes e servidores através de um canal *full-duplex* sobre um único *socket*.

Além do problema com interatividade entre a aplicação e o usuário, outro problema estava na execução do código do usuário. Originalmente, o CodeLive se

propunha a executar código em linguagem de programação Java, entretanto, nos trabalhos anteriores não houve a criação de uma ferramenta capaz de executar os códigos de maneira eficiente. Ademais, o contexto de aplicação e utilização inicial da ferramenta havia mudado: os professores dos referidos cursos da área de computação passaram a utilizar a linguagem de programação *Python* no ensino e prática de algoritmos e linguagens de programação. Para atender ao contexto da prática da programação, em si, foi utilizado o *framework Skulpt*, que executa códigos na linguagem de programação *Python*.

As ferramentas citadas (*Socket.io* e *Skulpt*) foram fundamentais no desenvolvimento deste trabalho: a primeira oferece uma API JavaScript simples, baseada em eventos que permite a comunicação entre o servidor e o cliente em tempo real. Por padrão, o mecanismo de comunicação do *framework* é o *WebSocket*, caso o navegador do usuário não dê suporte a essa tecnologia, ela recorre a *fallbacks*, como *Flash* e *Ajax* [ZIM 2013]. O *Skulpt*, por sua vez, é um *framework* que permite executar códigos *Python* diretamente no navegador do usuário sem a necessidade de *plug-ins* ou suporte no lado do servidor [GRAHAM 2016].

Ao fazer uso dos frameworks citados o projeto também teve sua arquitetura modificada, principalmente no que tange ao *back-end*. Inicialmente a solução foi desenvolvida utilizando a linguagem de programação PHP, porém, tanto o *Socket.io* quando o *Skulpt*, são implementados na linguagem de programação *JavaScript*. Logo, modificar o *back-end* da aplicação foi uma decisão razoável. O novo *back-end* da aplicação passou a ser executado sobre a plataforma *Node.js* (a plataforma torna possível a utilização de *JavaScript* no lado do servidor). Como consequência, a alteração trouxe o benefício de um código mais homogêneo, já que *back-end* e *front-end* (baseado no *framework AngularJs*) utilizavam o *JavaScript* como linguagem para o seu desenvolvimento.

2. Materiais e Métodos

Para que o objetivo do trabalho fosse alcançado, houve uma série de processos, os quais ocorreram em três etapas principais, ilustradas pela Figura 1.

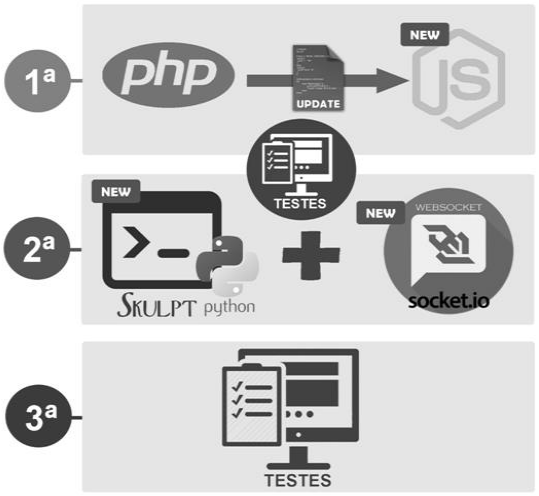

Figura 1 - Etapas de desenvolvimento do trabalho

Na primeira etapa foi feita a substituição da linguagem de programação utilizada no *back-end*, do servidor de aplicação e do *framework* de criação de serviços. Um dos fatores que levou a tais mudanças, é o fato de a linguagem *JavaScript* e o *Node.js* possuírem recursos e *frameworks* que são altamente compatíveis com a tecnologia *WebSocket*.

A segunda etapa foi de desenvolvimento das novas funcionalidades. Primeiro, foi incluído o *Skulpt*, visto que a aplicação já possuía o módulo com a função de receber, executar e validar o código do usuário ("módulo de desafio"). Após a inclusão do *Skulpt*, utilizou-se o *Socket.io* para implementar os métodos que fazem as notificações e atualizações em tempo-real de algumas partes da aplicação.

Na etapa final do desenvolvimento, realizou-se testes para validar o funcionamento das novas funcionalidades e comprovar a eficiência da aplicação como um todo. A metodologia adotada para tais testes, consistiu-se em utilizar a ferramenta e verificar se as ações feitas produziam o resultado esperado.

3. Resultados e Discussão

Nesta seção são apresentados os resultados obtidos após a atualização do CodeLive, o processo necessário para adaptar a ferramenta ao novo ambiente, a arquitetura da ferramenta, o funcionamento da ferramenta, os módulos existentes e a interação entre eles. Para guiar o processo de desenvolvimento e elucidar sobre cada módulo da aplicação e a interação existente entre eles, foi criada uma arquitetura, esta é ilustrada na Figura 2.

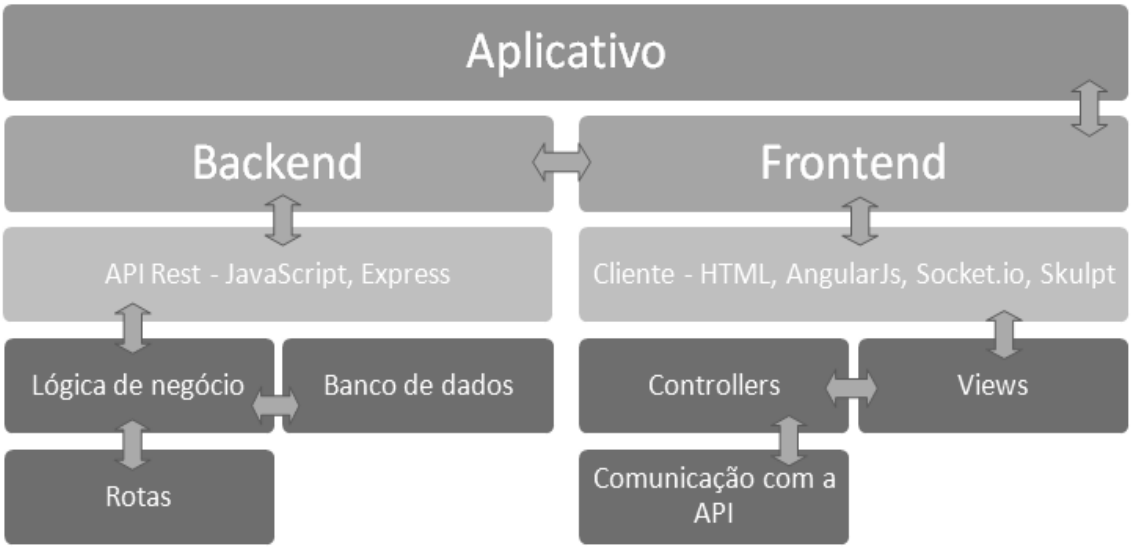

Figura 2 - Arquitetura do CodeLive

A arquitetura da Figura 2 possui onze módulos, os quais são apresentados em camadas (ou níveis). A comunicação entre os módulos é representada pelas setas bidirecionais, isso indica que a informação vai de um módulo ao outro e permite que uma informação de mais alto nível (módulo **Aplicativo**, por exemplo) seja processada em um módulo de baixo nível (módulo ***Controllers***, por exemplo) e retorne ao módulo inicial que proveu a informação.

De forma resumida o fluxo da informação começa no ***front-end*** e vai para o ***back-end***. Cada um dos módulos citados possui outros módulos (ou submodelos) que utilizam diversas tecnologias com o intuito de exibir, gerenciar e realizar a comunicação com outras partes da aplicação (no caso do ***front-end***). Para os módulos contidos no ***back-end*** a principal tarefa é a de implementar métodos que fazem o acesso ao banco de dados, e com isso, permitir a criação de serviços (cadastrar um usuário, por exemplo) que podem ser acessados/consumidos pelo o ***front-end.***

3.1. Comunicação via WebSocket

Apesar da comunicação (ou acesso aos recursos) entre o *front-end* e o *back-end* ocorrer através de rotas (URLs que dão acesso aos módulos presentes no *back-end*), o acesso aos recursos em tempo-real construídos com a tecnologia *WebSocket*, utiliza outro meio para troca de dados. Essa comunicação ocorre a partir de eventos, sendo estes identificados por um nome e disparados quando algo acontece no lado do cliente ou do servidor.

Front-end:

```
14      $scope.login = function (usuario) {
15          if ($scope.usuario.Senha != undefined && $scope.usuario.NomeExibicao != undefined) {
16              $http.post(host + "login", JSON.stringify({ usuario: usuario })).success(function (data) {
17                  if (data != "") {
18                      socket.emit("novoUsuarioLogado", data);
```

Back-end:

```
23  io.on('connection', function (socket) {
24    socket.on('novoUsuarioLogado', function (data) {
25      socket.broadcast.emit('novoUsuarioLogado', data.NomeExibicao + " entrou no jogo!");
26    });
27  });
```

Front-end:

```
106       socket.on('novoUsuarioLogado', function (msg) {
107           notification('info', 'AVISO', msg);
108       });
```

Figura 3 - Exemplo de troca de mensagens via WebSocket

A Figura 3 ilustra uma série de funções *WebSocket* utilizadas para notificar um usuário sobre a entrada de um outro usuário na aplicação. Essas funções utilizam o *framework Socket.io* tanto no lado do cliente quanto no lado do servidor, e o código na Figura 3 demonstra o que é necessário para que essa troca de dados seja possível. Nesse caso, o *front-end* possui um método que envia um dado, e outro que aguarda um dado (respectivamente, linha 18 da primeira imagem e linha 106 a terceira imagem). No *back-end* o processo é similar, salvo o fato da mensagem ser enviada a todos os *sockets* conectados, exceto ao *socket* que enviou o dado (faz isso usando o método "*broadcast*", linha 25), em outras palavras o usuário que entrou no jogo não é notificado, somente os demais usuários que já estavam conectados.

Utilizando de métodos como os ilustrados na Figura 3, foi possível construir um mecanismo eficiente de comunicação em tempo-real, voltado principalmente para notificar os usuários sobre as interações ocorridas na aplicação e atualizar partes da interface quando necessário. Vale ressaltar que o *WebSocket* permite o tráfego de dados

binários, tornando possível o envio não somente de mensagens de texto, mas também de dados mais complexos como objetos e listas de objetos.

3.2. Conhecendo o CodeLive

Para um melhor entendimento sobre o funcionamento da ferramenta é importante conhecer qual o funcionamento geral da mesma, as regras utilizadas, o que o usuário consegue ou não fazer e quais metas devem ser atingidas para que o usuário avance no jogo.

3.2.1. Público alvo

O CodeLive é uma ferramenta de utilização livre voltada para o ensino e prática de programação, logo, seu público alvo são estudantes e professores. Porém, ela não se limita somente a essas pessoas, também sendo destinada a qualquer um que se interesse por aprender e praticar programação.

3.2.2. Funcionamento do jogo

O jogo consiste em resolver desafios, esses desafios são exercícios de programação constituídos de um título, enunciado entre outros dados. A resolução dos desafios é necessária para que o usuário avance no jogo, assim, esse avanço é medido com três dados principais: DF (domínio da força), Classe (podem ser: Novato, Padawan, Cavaleiro Jedi, Mestre Jedi e Mestre do Conselho) e Nível (nível em que o jogador se encontra dentro de uma classe, “Mestre Jedi nível 3”, por exemplo). Essas informações são vistas pelos outros usuários e mostram o quão avançado um usuário está no jogo.

Ao realizar o cadastro o usuário inícia o jogo na classe “Novato”, “Nível 1” e com 50 créditos, para que um jogador evolua, ou seja, aumente sua Classe e/ou Nível, é necessário vencer os desafios e com isso ganhar pontos de DF. Além dos pontos de DF, ao vencer um desafio, o usuário também ganha créditos e eventualmente pode vir a ganhar Recompensas. Os créditos são utilizados para comprar os desafios, já as Recompensas são prêmios que o jogador ganhar por realizar algumas ações, como por exemplo, chegar ao nível dois da classe Padawan. Caso os creditos do usuário acabem, este pode acessar o módulo de “desafios comprados” e utilizar a opção de “treino” para conseguir mais créditos e seguir no jogo.

Para complementar as informações descritas pela presente seção, vale ressaltar que, por causa da gamificação, há ainda regras e *feedbacks*, que contribuem na organização do ambiente bem como orientam o usuário acerca do seu desempenho no jogo. Umas das regras é a de requisito de um certo nível de experiência (classe e nível mais elevados) para a criação de desafios. Sobre os *feedbacks*, eles estão presentes no módulo de **perfil do usuário** (Figura 4). Neste módulo o usuário encontra o "*ranking* dos melhores jogadores", o quantitativo de desafios ganhos e perdidos, o número de desafios ganhos por nível de dificuldade, as recompensas existentes e as que já foram ganhas entre outras informações.

3.2.3. Módulos do CodeLive

O CodeLive é composto por diversos módulos com as mais diversas funcionalidades. Esses módulos são: o módulo de login, o módulo de cadastro de usuário, o módulo de

perfil, o módulo de visualização e compra de desafios, módulo de cadastro de desafio, o módulo de responder desafios e o módulo dos desafios comprados. Os tópicos a seguir descrevem os módulos citados.

3.2.4. Módulo de login

O módulo de login é a primeira tela que o usuário tem contato ao acessar o sistema. Sua principal função é fazer a autenticação do usuário, solicitando as informações "Nome de exibição" e "Senha". Após preencher o formulário com os dados requeridos, o usuário seleciona a opção "Entrar" para submeter as informações e aguarda o resultado. Se os dados estiverem corretos o usuário é redirecionado para o módulo de Perfil do usuário, caso contrário, uma mensagem é exibida informando um erro nos dados.

3.2.5. Módulo de cadastro de usuário

O acesso a esse módulo se encontra na tela de Login, através da opção: "Crie sua conta". A função desse módulo é fazer um registro do usuário, para que o mesmo possa acessar o sistema. Os dados exigidos para o cadastro do usuário, são: "Nome completo", "E-mail", "Nome de exibição", "Senha" e uma "Confirmação da senha". Além dos dados citados o usuário pode selecionar um avatar. Se nenhum avatar for escolhido o sistema adota o avatar padrão que pode ser alterado a qualquer momento no módulo de Perfil do usuário, junto com as demais informações do usuário.

3.2.6. Módulo de perfil

O módulo de perfil do usuário é a primeira tela exibida após a autenticação do usuário. Para um melhor entendimento, esse módulo é dividido em três partes. A primeira parte do perfil é responsável por mostrar as **informações gerais do usuário**. As informações presentes nessa tela dizem respeito ao desempenho do usuário dentro do jogo, também há um *ranking* dos melhores jogadores como ilustra a Figura 4.

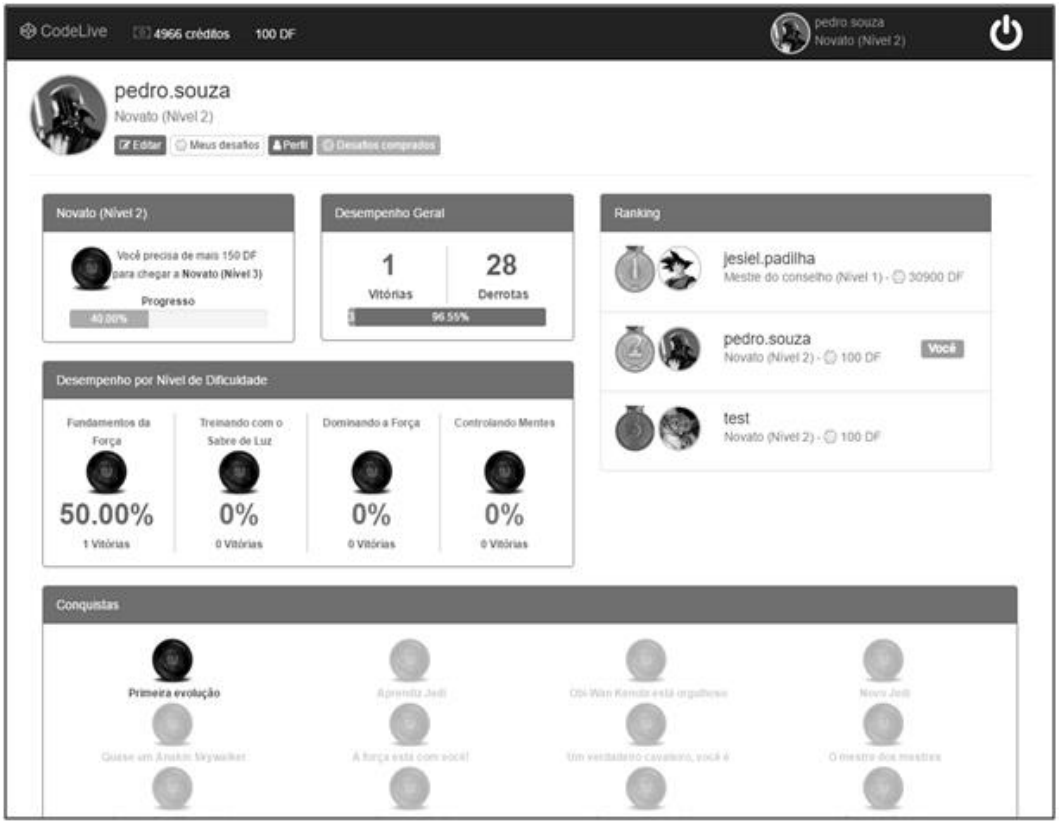

Figura 4 - Tela de perfil com as informações gerais do usuário

A segunda parte do perfil se trata da **edição dos dados do usuário**, sendo criada para que o usuário possa alterar os dados informados no momento do cadastro. A última parte do módulo de **Perfil do usuário** é a de "**Gerenciar meus desafio**s", sendo está destinada ao gerenciamento dos desafios criados pelo usuário.

3.2.7. Módulo de visualização e compra de desafios

O módulo de visualização e compra de desafio lista todos os desafios criados e publicados. Um desafio é exibido no formato de um painel que possui diferentes cores, sendo que cada cor representa o grau de dificuldade para resolução daquele desafio. O painel também contém outras informações como: título, usuário que criou o desafio, valor do desafio entre outros. Após clicar em um dos painéis, uma tela é exibida com mais detalhes do desafio e a opção para comprar o desafio em questão.

3.2.8. Módulo de cadastro de desafio

Para que haja desafios disponíveis no módulo descrito no tópico anterior, é necessário que alguém os cadastre. Para isso, existe o módulo de castro de desafio. Ele é composto por um formulário com diversas opções, cujas principais e obrigatórias são: Título do desafio, Tags, Entradas, Saídas, Nível de dificuldade, Descrição resumida e Enunciado do desafio.

É importante ressaltar que uma resposta a um desafio é validada pela ferramenta através da análise de suas entradas e saídas, portanto, para que isso seja possível, as entradas e saídas devem ser inseridas na ordem da execução em que serão executadas

pelo *Skulpt* no módulo de **Responder desafio**. Tanto as entradas quanto as saídas são inseridas em um campo de texto, e podem ser valores numéricos ou uma cadeia de caracteres (um texto).

Como há uma ordem de execução das entradas e saídas, estas são interpretadas como um documento, ou seja, linha à linha, isso influencia no momento do cadastro de desses dados. Assim o usuário é obrigado a realizar uma quebra de linha para separar cada entrada e saída na ordem que se deseja executar cada uma delas.

3.2.9. Módulo de responder desafios

O módulo de responder desafios é um dos módulos mais importantes do sistema, pois é a área disponível para responder os desafios adquiridos pelos usuários, permitindo assim o avanço do mesmo no jogo.

Figura 5 - Tela para responder os desafios

A Figura 5 ilustra a tela do módulo de responder desafios, a mesma é composta por:

- um editor de texto (local no qual o usuário insere o código que será utilizado posteriormente para responder o desafio adquirido);
- um relógio (para informar o usuário acerca do tempo restante para responder um desafio, caso exista);
- opções de gerenciamento do editor (os mesmo servem para mudar alterar o tamanho e o tema do editor);
- informações do desafio (compostas pelas informações do autor do desafio, bem como as recompensas que o usuário ganhará se responder o desafio corretamente e a descrição do que se deve fazer); e
- botões para executar o código do editor, limpar o editor e confirmar a resposta.

Uma das interações possível pelo presente módulo é o *feedback* que o usuário tem ao executar o código: caso ocorra algum erro este é apresentado no console abaixo

do editor de texto na cor vermelha, informando por exemplo, que uma variável não existe.

Deve-se esclarecer que há duas execuções possíveis neste módulo e que o *feedback* citado só é possível em uma delas. A primeira forma de execução ocorre ao selecionar a opção "Executar", após essa ação, uma caixa de diálogo *JavaScript* é exibida, a mesma serve para que o usuário digite um valor qualquer, sendo esta exibida sempre que houver a existência da função *Python* "input()" (função utilizada para armazenar um valor informado pelo usuário).

A segunda forma de executar o código não permite intervenção/interação do usuário e ocorre ao selecionar a opção "Confirmar". Este modo de execução ocorre em um *controller* e realiza a validação do código do usuário através de uma análise das entradas e saídas pré-cadastradas do desafio.

3.2.10. Módulo de desafios comprados

Este módulo foi criado para que o usuário pudesse ver os desafios dos quais ele já participou e, com isso, ter a possibilidade de melhorar suas habilidades, já que este módulo oferece uma área de treino com os desafios adquiridos.

Figura 6 - Tela de detalhes de um desafio comprado

A Figura 6 exibe os detalhes de um desafio em que o usuário já participou. Nota-se que são exibidas as principais informações de um desafio, como por exemplo: Autor do desafio, Descrição do desafio, Enunciado, Tempo para responder o desafio e o Pré-código, caso exista. Outra informação de grande relevância são as Estatísticas, ou seja, o desempenho do usuário no desafio em questão, com esse dado é possível saber o

número de vezes que o usuário participou daquele desafio além da quantidade de vitórias e derrotas.

Ainda nesse módulo, o usuário pode ver o código utilizado nas diversas vezes que respondeu esse desafio, tendo também a possibilidade de melhorar o código produzido através da área de treino. Tal área fornece um ambiente similar ao utilizado no módulo de **responder desafios**, tornando possível a execução do código e até mesmo, salvar o código produzido. Ao utilizar a área de treino o usuário é recompensado com 1 crédito e poderá visualizar os códigos salvos na página de detalhes (Figura 6).

4. Conclusão

O presente trabalho teve a tarefa de complementar os recursos do CodeLive a fim de torná-lo mais eficiente no que tange à execução do código do usuário, e mais interativo, fornecendo informações em tempo-real para o usuário, sem que houvesse a necessidade do mesmo realizar alguma ação. Tais objetivos foram alcançados utilizados as ferramentas *Socket.io*, *Skulpt*, *Node.js* e *AngularJs*. Com os recursos disponibilizados por essas ferramentas foi possível utilizar a linguagem de programação *JavaScript* tanto no *front-end* quanto no *back-end* da aplicação, construir as funções de notificação e atualização em tempo-real e executar o código *Python* inserido pelo usuário, viabilizando também o processo de validação de um desafio pela própria ferramenta. Elas também contribuíram para a criação da interface do usuário e dos serviços presentes na aplicação.

Dentre as diversas contribuições feitas pelo presente trabalho estão as mensagens de notificação quando um usuário entra na aplicação, quando um novo desafio é criado, quando um usuário responde um desafio que outro usuário também respondeu (o autor do desafio também é notificado), entre outras interações em tempo-real. Todas essas mensagens são feitas em tempo-real, não importando o módulo, ou seja, a parte da aplicação que o usuário esteja acessando, sendo necessário apenas que o usuário esteja logado. Nesse sentido, a ferramenta passa a interagir com o usuário de forma autônoma, poupando o mesmo de deslocamentos desnecessários para obter informações e fornecendo *feedbacks* acerca do uso da aplicação por outros usuários.

As demais contribuições foram relacionadas à criação e atualização de alguns módulos. Um dos módulos criados foi o de “desafios comprados”, que permite ao usuário ver os desafios adquiridos por ele, além de possibilitar que este treine utilizando um desses desafios e como isso aumente sua experiência para responder desafios futuros. Já no módulo de perfil houve uma atualização, e agora há painéis que fornecem dados sobre o desempenho do usuário no jogo, informando por exemplo, quantas vitórias e quantas derrotas o usuário teve.

Para trabalhos futuros a interação entre os demais usuários é um ponto de extrema importancia. A criação de uma ferramenta de Chat pode ser o primeiro passo nessa interação, isso possibilitaria aos usuários conversarem entre si e trocarem experiências acerca de desafios resolvidos, por exemplo. Outro recurso bastante relevante e descrito inclusive por Cardoso (2015) em seu trabalho, é o de duelo entre os usuários: nesse módulo dois usuários iriam competir entre si, sendo que o mais rápido a responder um desafio de forma correta seria dado como vencedor. Tais melhorias

podem contribuir na disseminação do conhecimento, além de promoverem a ferramenta e torna-la mais eficiente e democrática, possibilitando que a mesma alcance um maior número de pessoas, as quais podem elevar o nível de conhecimento fornecido pela ferramenta.

5. Esclarecimentos

Esta seção tem o objetivo de elucidar acerca da metafora utilizada no presente trabalho. O mesmo utiliza o termo "batalha espacial", porém nota-se que a parte visual da aplicação assim como alguns aspectos da mecânica do jogo (classe do jogador, por exemplo), remetem a série de filmes "Star Wars", de fato os elementos citados foram baseados no universo de "Star Wars", porém o termo não foi utilizado, já que para isso se faz necessário a autorização expressa da Disney, empresa detentora dos direitos autorias da marca.

Referências Bibliográficas

ALVARENGA, Sean Carlisto de. Tecnologias Push e uma arquitetura de notificação para cidades inteligentes. 2013. 62 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Estadual de Londrina, Londrina, 2013. Disponível em: <http://www.uel.br/cce/dc/wp-content/uploads/TCC-SeanAlvarenga-BCC-UEL-2013.pdf>. Acesso em: 19 jan. 2016.

CARDOSO, Djonathas Carneiro. CODE LIVE: Gamificação de um ambiente virtual de programação. 2015. 85 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas, 2015.

GRAHAM, Scott. Skulpt: Python. Client side. Disponível em: <http://www.skulpt.org/>. Acesso em: 23 nov. 2016.

MULLER, Gabriel L. HTML5 WebSocket protocol and its application to distributed computing. 2013. 63 f. Dissertação (Mestrado) - Curso de Ciência da Computação, School Of Engineering, Computational Software Techniques In Engineering, Cranfield University, Swindon, 2014. Disponível em: <http://arxiv.org/pdf/1409.3367v1.pd>. Acesso em: 15 jan. 201

PADILHA, Jesiel Souza. CodeLive: Interatividade com o usuário em tempo-real e Interpretação e execução do código do usuário. 2016. 59 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas, 2016.

ROSS, Keith W.; KUROSE, James F. Redes de Computadores e a Internet: Uma Abordagem Top-Down. 6. ed. São Paulo: Pearson Education - Br, 2013. 634 p

ZIM, Joe. Conectando no Socket.IO: o básico. 2013. Disponível em: <http://imasters.com.br/tecnologia/redes-e-servidores/conectando-no-socket-io-o-basico/?trace=1519021197&source=single>. Acesso em: 12 jul. 2016.