CodeLive: reformulação da arquitetura e implementação

Karoline Dias Barreto[1], Jackson Gomes de Souza[1], Fabiano Fagundes[1]

[1] Ciência da Computação - Centro Universitário Luterano de Palmas (CEULP/ULBRA)
Avenida Teotônio Segurado, CEP 77.019-900 – Palmas – TO – Brasil

karol.db1230@gmail.com, jgomes@ceulp.edu.br, thilfa@gmail.com

Resumo. *Um ambiente online de execução de código que auxilia no aprendizado dentro e fora de aula tem como necessidade garantir a segurança do ambiente de execução. A partir dessa compreensão, este trabalho adotou a proposta que a execução do código do usuário ocorresse em um ambiente virtualizado, garantindo, dessa forma, a seguridade do software e evitando danos. O presente trabalho objetiva o desenvolvimento do CodeLive, com a utilização da linguagem de programação Python, o framework Django e o software Docker, que constitui uma camada intermediária no software e possibilita a criação de ambientes isolados para a execução dos códigos-fontes em Python informados por alunos.*

1. Introdução

As práticas de ensino têm passado por adaptações, tendo como objetivo incentivar, despertar interesse e auxiliar no aprendizado dentro e fora da sala de aula. Ambientes Virtuais de Aprendizagem (AVA) surgem como uma dessas adaptações, possibilitando uma interação professor-aluno além do contexto da sala de aula.

Nessa perspectiva, foi desenvolvido o CodeLive, ambiente gamificado de programação que utiliza elementos de gamificação juntamente com um enredo de universo de batalha estelar, de modo a obter uma plataforma didática, divertida e de constante crescimento, já que a mesma permite que os jogadores acrescentem novos conteúdos (CARDOSO et al, 2016).

O CodeLive teve seu desenvolvimento na plataforma web por Padilha (2016), utilizando a plataforma *NodeJS* e recursos de programação como *WebSocket*, viabilizando a execução do código-fonte *Python* informado por usuários diretamente no *browser*. Nesse sentido, o CodeLive pode ser utilizado como ferramenta de suporte a um curso de programação, permitindo que estudantes respondam desafios de programação utilizando a linguagem *Python*.

Embora esse formato de execução do código-fonte *Python* no *browser* seja adequado, há limitações de execução e recursos, daí a necessidade de executar o código-fonte no servidor web. Quando há uma decisão nesse sentido surgem problemas como os relacionados à segurança do servidor web, como o que ocorreria se um trecho do código fosse malicioso ao ponto de apagar arquivos ou tentasse acessar uma funcionalidade de acesso protegido.

Questões como essa podem ser resolvidas utilizando recursos de virtualização de servidores. Por exemplo, ao executar um código-fonte em um ambiente virtualizado, a máquina virtual é isolada do *host*, permitindo maior controle sobre o acesso a recursos do mesmo. O *Docker* é um tipo de tecnologia que implementa a virtualização de forma diferenciada, pois, como afirma Diedrich (2015), este possui recursos isolados que utilizam bibliotecas de kernel em comum (entre *host* e *container*), diferente de ambientes de virtualização tradicionais, que possuem um Sistema Operacional completo e isolado.

Dentro desse contexto, o presente trabalho apresenta o desenvolvimento de uma versão do *CodeLive* que executa código-fonte *Python* informado pelos estudantes em um ambiente protegido com virtualização.

ISSN: 2447-0767

2. Reformulação do CodeLive

O CodeLive, ambiente gamificado para aprendizado de programação, teve como objetivo que a interpretação e execução do código que o usuário inserisse no ambiente ocorresse em tempo-real, diretamente no browser (PADILHA, 2016). Dessa forma, a pessoa que estivesse utilizando o CodeLive se sentiria mais engajado, tanto pela gamificação do ambiente quanto pela interação proporcionada pelo software.

Entretanto, o CodeLive desenvolvido por (PADILHA, 2016) não teve o propósito de adotar práticas de segurança na execução do código-fonte informado pelo estudante, motivo pelo qual foi necessário repensar sua arquitetura e programação, de maneira que garantisse a segurança.

Para que fosse possível analisar as diferenças entre o CodeLive desenvolvido por (PADILHA, 2016) e o CodeLive reformulado e desenvolvido no presente trabalho, a comparação entre as arquiteturas dessas versões do CodeLive foi realizada, como apresentam as Figuras 1 e 2.

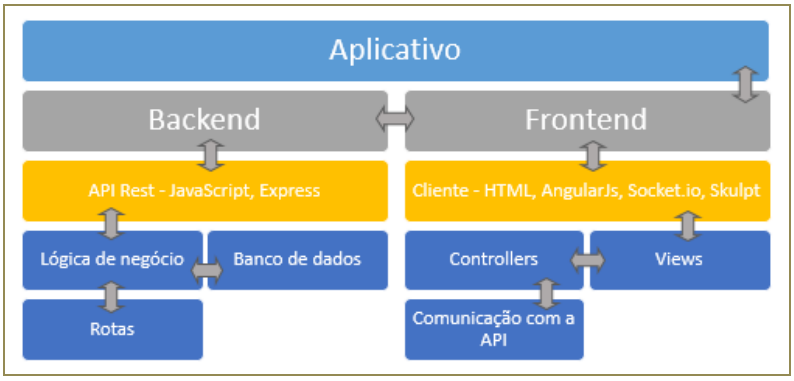


Figura 1: Arquitetura do CodeLive
Fonte: (PADILHA, 2016)

Na Figura 1 pode-se notar que o ambiente é composto por uma camada da aplicação (aplicativo) que faz conexão com a camada *Frontend*. Esta, por sua vez, faz conexão tanto com a camada *Backend* quanto com as tecnologias de alto nível (*HTML*, *AngularJs*, *Socket.io* e *Skulpt*). Por fim, a camada faz ligação com a camada *Views*, que faz ligação com a *Controllers* que, por fim, faz a ligação com a camada *Comunicação com a API*.

Do lado do *Backend* há a conexão com camada *API Rest*, na qual são utilizadas as tecnologias de baixo nível: *JavaScript* (apesar de ser considerada uma linguagem de alto nível foi utilizada nas duas camadas, *Frontend* e *Backend*) e o *Express*. A camada *API Rest* faz a ligação com a *Lógica de Negócio*, que faz ligação tanto com o *Banco de dados* quanto com a camada de *Rotas*. A Figura 2 apresenta a arquitetura reformulada do CodeLive, resultado não só dos objetivos que já foram apresentados, mas também de mudança de plataforma de execução.

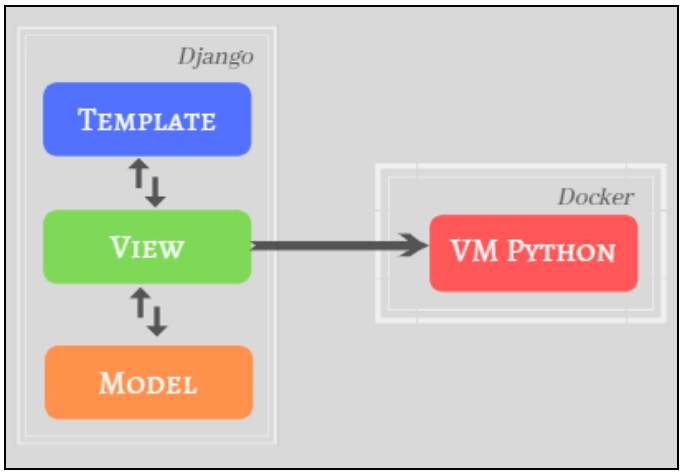


Figura 2: Arquitetura Reformulada do CodeLive

A arquitetura reformulada do CodeLive conta com três componentes básicos: *Django*, *Docker* e *VM Python* (máquina virtual do *Python* que é criada dentro do *Docker*). A *view* faz conexão direta à máquina virtual do *Python*, garantindo que a execução do código inserido pelo usuário aconteça em um ambiente isolado.

Os recursos de gamificação definidos anteriormente no ambiente desenvolvido por (PADILHA, 2016) foram retirados do escopo da reformulação do CodeLive, priorizando a utilização de novas tecnologias para assegurar a segurança do ambiente.

3. Materiais e Métodos

Para o desenvolvimento do presente trabalho foram utilizados os seguintes recursos:

- a linguagem de programação *Python*, que é de código aberto, orientada a objetos, interpretada e de alto nível;
- *Django*, um *framework web* de código aberto para aplicações web que utilizam o padrão de arquitetura MVC (*Model-View-Controller*);
- *Docker*, da empresa *Docker Inc*., é uma alternativa de virtualização, pois fornece uma camada adicional de abstração e automação de virtualização de nível de sistema operacional. O mesmo oferece contêineres, sendo estes ambientes isolados. (GOMES; SOUZA, 2015).
- o login social é um pacote para o *Django* que possibilita que o usuário realize o login em uma determinada plataforma a partir de contas que ele já possui em outros serviços (como *Google* e *GitHub*).
- *Markdown* é uma linguagem de marcação leve e possui uma sintaxe de formatação de texto simples com o objetivo de ser convertida em *HTML* e outros formatos. O pacote *Markdown2* é uma implementação *Python* do *Markdown* e foi utilizado transformar para áreas de texto escritas em *Markdown* para *HTML*;

Para execução do presente trabalho, o processo foi dividido em sete etapas (Figura 3).

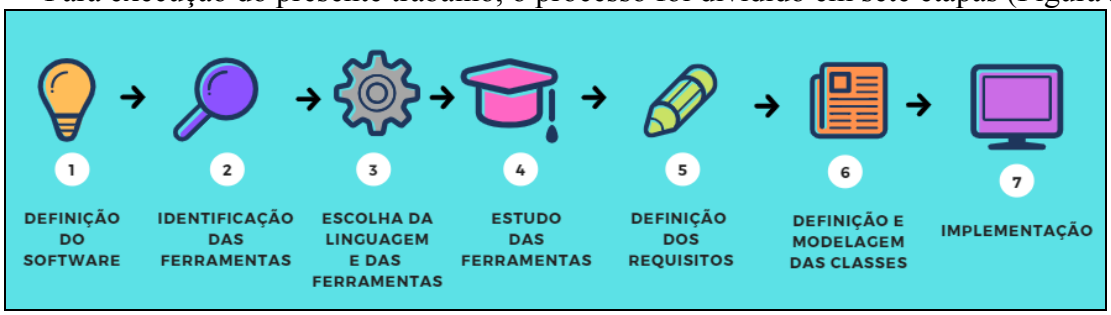


Figura 3: Processo de Desenvolvimento

O processo de desenvolvimento (Figura 3) foi iniciado com a definição do software e suas características (etapa 1), passando para a identificação de ferramentas que seriam necessárias para o desenvolvimento do software (etapa 2) e escolha da linguagem (*Python*) e das ferramentas que seriam utilizadas, sendo estas *Django* e *Docker* (etapa 3), para realizar o estudo sobre essas ferramentas (etapa 4) e, a partir disso, definir os requisitos do software (etapa 5), definir e modelar as classes do ambiente (etapa 6) e, por fim, implementar o software (etapa 7).

4. Resultados

Para o entendimento das ações que serão realizadas no software (tanto por usuários quanto por componentes do sistema) a Figura 4 ilustra a visão geral do software, sendo composta por uma interface administrativa, na qual os usuários poderão ter acesso às funcionalidades; uma camada intermediária baseada no *Docker*, que faz ligação com a interface administrativa do *Django* e com o terceiro componente; e a máquina virtualizada do *Python*.

ISSN: 2447-0767

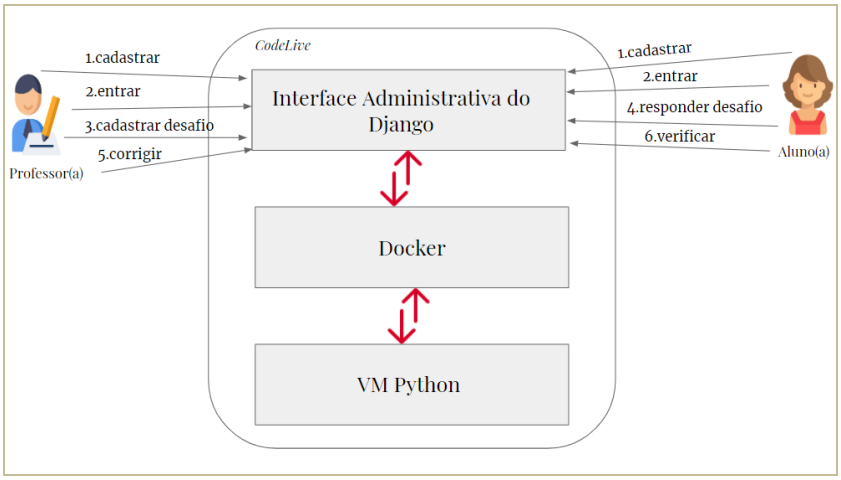


Figura 4: Visão Geral do Ambiente

Os usuários do software são representados por dois atores principais com funcionalidades específicas: Professor e Aluno. O Professor se cadastra no sistema, entra no sistema, cadastra desafio (incluindo casos de teste que serão utilizados na correção automática das respostas) e corrige a resposta do aluno. O aluno se cadastra no sistema, entra no sistema, responde ao desafio e verifica o resultado da correção da resposta. Há ainda um terceiro ator, o Administrador, que associa papeis (grupos) aos usuários e indica qual é aluno ou professor.

A camada *Docker* do ambiente faz o intermédio entre a interface administrativa do *Django* e a máquina virtual do *Python*, possibilitando a criação de um ambiente isolado para execução de código fornecido pelo usuário, que se dá no momento da correção de uma resposta a um desafio. Para cada correção de resposta, uma imagem da máquina virtual é criada e é dentro da mesma que há a execução o código-fonte *Python*.

O presente trabalho foi implementado utilizando a interface administrativa que o *Django* oferece. O django-admin é uma interface administrativa do *Django* que atua nos metadados dos *models* podendo, dessa maneira, fornecer recursos de gerenciamento dos dados, como listar, cadastrar, editar e excluir.

A funcionalidade **Criar conta de usuário** é atribuída aos dois atores do sistema (Professor e Aluno). A Figura 5 apresenta o formulário de cadastro, contendo um campo para inserção do nome de usuário (*Usuário*) e outro campo para inserção da senha (*Password*).

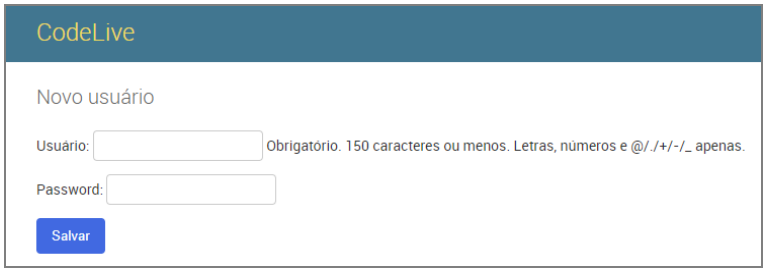


Figura 5: Tela de Cadastro de Usuário

Após realizar o cadastro, o usuário acessa a **tela de login**, que é apresentada na Figura 6. Para realizar o login, o usuário deve utilizar os dados que ele utilizou para criar sua conta de usuário, preenchendo os campos *Usuário* e *Password* ou utilizar o login social, via *Google* ou *GitHub*. Após realizar o login, com autenticação bem-sucedida, o usuário é direcionado pelo sistema para a tela inicial da interface administrativa que, como já informado, apresenta funcionalidades conforme suas permissões.

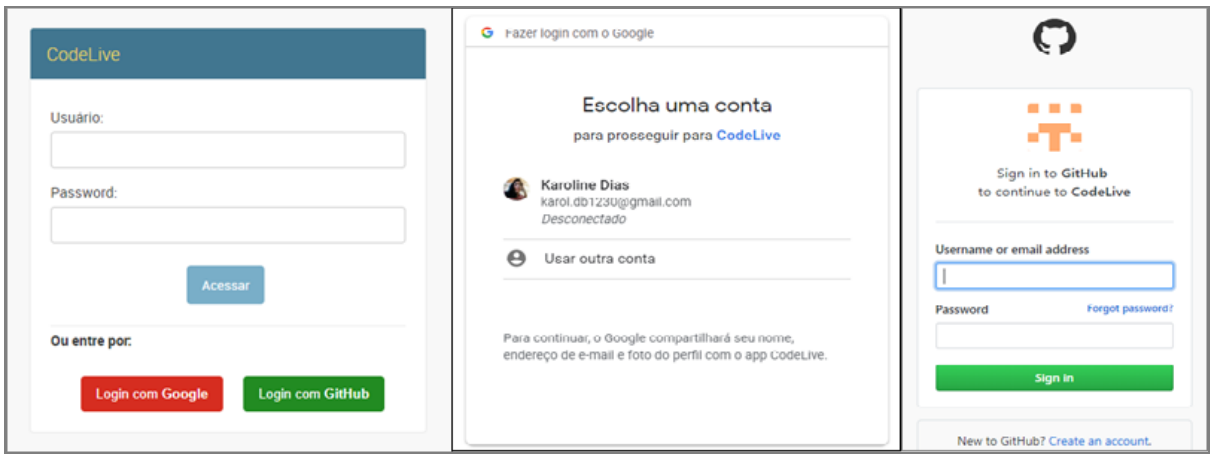


Figura 6: Telas de Login

A funcionalidade **Cadastrar desafio** é atribuída apenas para o usuário do tipo Professor. As informações para cadastrar o desafio, como apresenta a Figura 7, são: título do desafio; a lista de tags, que são cadastradas previamente ou ao clicar no botão "+" ao lado da listagem das mesmas; enunciado; descrição do desafio; um pré-código, para auxiliar o aluno no momento em que o mesmo for responder o desafio; o nível de dificuldade, que pode ser cadastrado previamente ou através do clique no ícone "+" ao lado; está publicado; e arquivo, com informações para auxiliar no desafio, que poderá ser baixado pelo aluno.

Os campos "enunciado", "descrição" e "pré-código" podem receber conteúdo em formato *Markdown*. A utilização de *Markdown* fornece uma alternativa à utilização de um editor de textos com formatação – obviamente, o professor que utilizar esse recurso deve conhecer a formatação.

Figura 7: Tela de Cadastro de Desafio

Além de cadastrar as informações do desafio, o Professor cadastra os Casos de Teste (Figura 8), que são situações às quais os códigos-fonte fornecidos como resposta pelos Alunos são submetidos. O Caso de teste é representado pelo par (entrada, saída): a entrada fornecida para o código-fonte do Aluno e a saída esperada para o código-fonte do Aluno ao receber a entrada. O Professor pode cadastrar quantos casos de teste julgar ser necessário para aquele desafio.

ISSN: 2447-0767

Figura 8: Tela de Cadastro dos Casos de Teste

A funcionalidade **Responder a um desafio** é atribuída ao usuário do tipo Aluno. Para responder um desafio o usuário Aluno acessa o item "Desafios Cadastrados". Esse item redireciona o aluno para a tela onde são apresentados todos os desafios que foram cadastrados pelos professores, apresentando o título, o autor, o enunciado e o nível de dificuldade referentes ao desafio, como mostra a Figura 9.

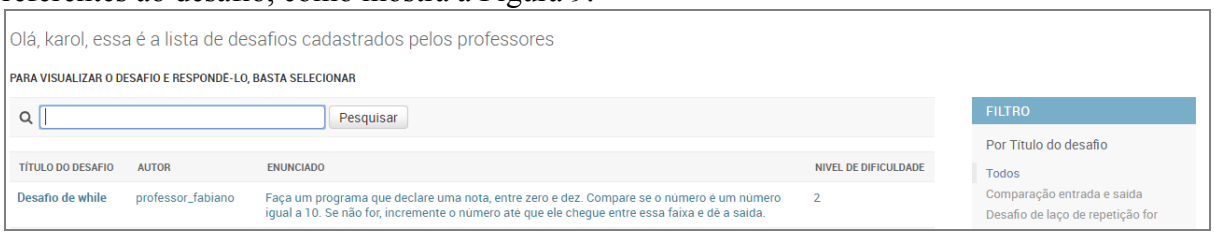


Figura 9: Tela de Listagem dos Desafios Cadastrados

O Aluno, dessa forma, seleciona um desafio da lista e isso vai levá-lo para a tela do desafio selecionado (Figura 10) contendo as informações do desafio, como: o autor do desafio, a data no qual foi cadastrado, o nível de dificuldade do desafio, o enunciado, a descrição e o pré-código. Caso o Professor tenha realizado *upload* de um arquivo o mesmo fica disponível para o aluno fazer *download*.

Figura 10: Tela de Visualização do Desafio

É na tela de visualização do desafio que o aluno responde o desafio. Ao final da descrição do desafio a área para que o aluno insira a resposta para aquele desafio é apresentada, como apresenta a Figura 11. O aluno pode modificar a sua resposta e apagá-la também através dessa tela.

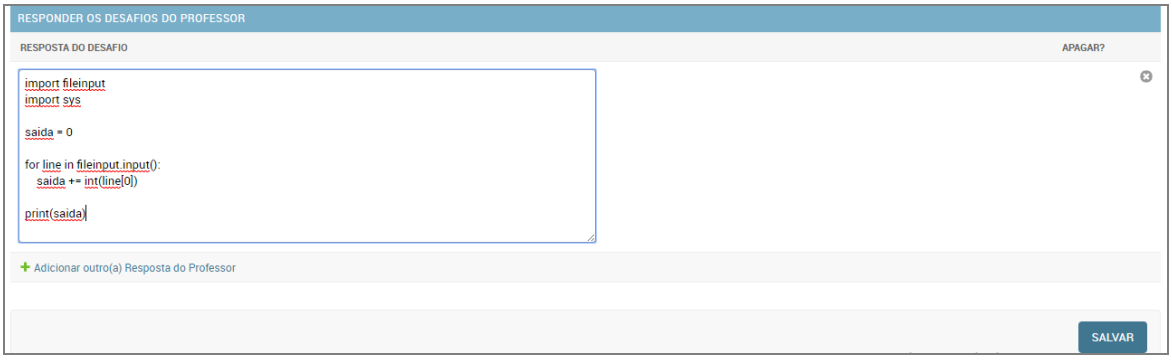


Figura 11: Campo para Responder Desafio Selecionado

Para **Corrigir uma resposta** o usuário (Professor) acessa o item "Respostas" da tela inicial do ambiente, o que o leva para a lista de respostas (referentes aos desafios cadastrados

por ele) enviadas pelos Alunos, contendo: o desafio, o usuário e a data de resposta, como mostra a Figura 12.

Figura 12: Tela de Listagem das Respostas do Desafio (Professor)

O Professor, então, seleciona a resposta (uma ou mais), clica na ação "Corrigir respostas selecionadas" e clica no botão ir, como a Figura 13 apresenta.

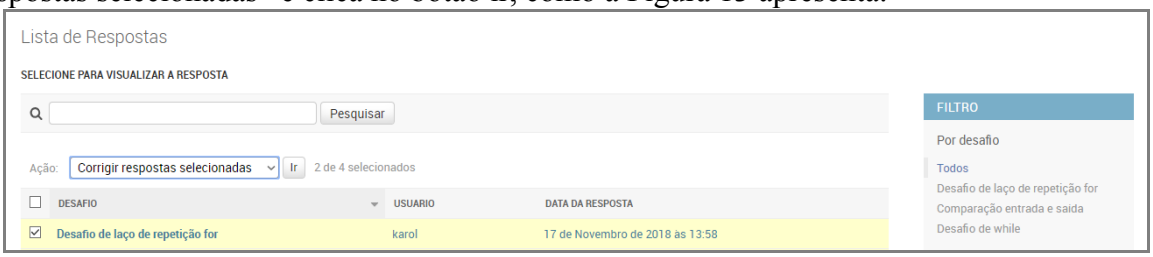


Figura 13: Seleção de Resposta para Correção

Depois dessa ação uma mensagem é exibida na tela informando que as respostas selecionadas foram corrigidas, como apresenta a Figura 14.

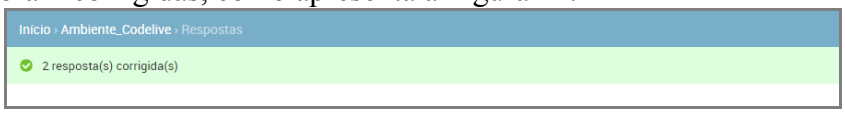


Figura 14: Tela da Listagem das Respostas com Mensagens de Correção

Para **Visualizar a correção de uma resposta**, o Aluno acessa o item "Respostas", que o leva para a tela de listagem das respostas (apenas as dele), como apresenta a Figura 15.

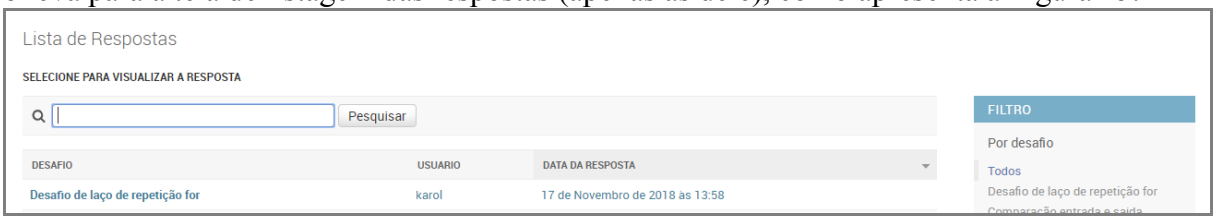


Figura 15: Tela da Listagem das Respostas (Aluno)

A partir disso, o Aluno seleciona a resposta da qual ele pretende visualizar a correção. Isso o leva à tela que apresenta as informações da resposta, sendo estas o desafio ao qual a resposta se refere, a data em que foi respondida e o código da resposta; e uma tabela com os dados da correção, contendo a data da correção da resposta, a pontuação que ele obteve com a resposta e um *link* para mais informações, como apresenta a Figura 16.

ISSN: 2447-0767

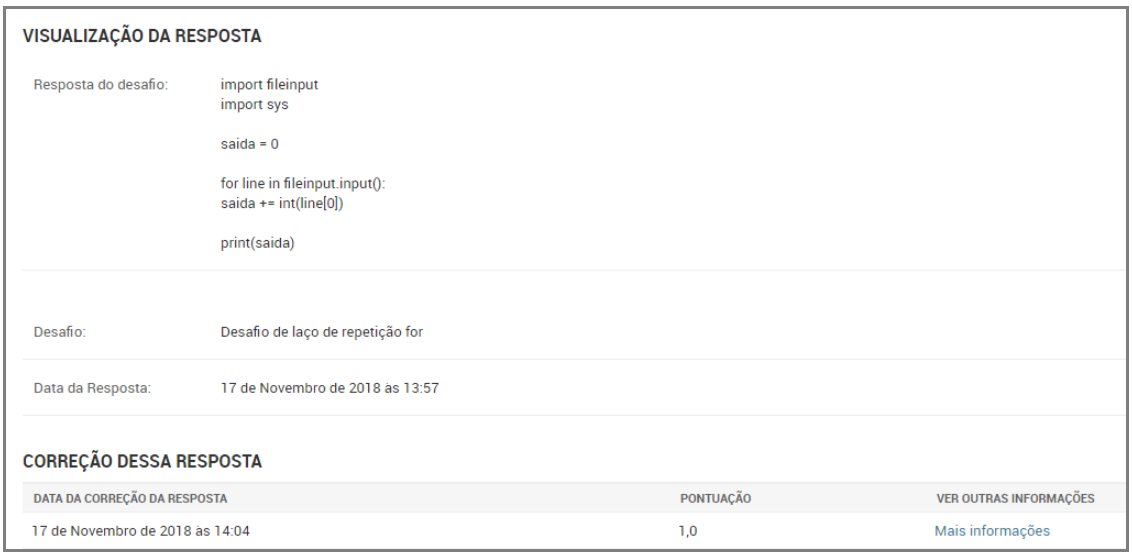


Figura 16: Tela da Visualização da Resposta Selecionada

A pontuação da resposta indica se o programa enviado pelo aluno para responder ao desafio está correto, de acordo com os casos de teste. Para visualizar outras informações sobre essa correção o aluno acessa o *link* "Mais informações" que o leva à tela de visualização da correção completa. A tela, por sua vez, apresenta as informações da correção, como (Figura 17):

- a data em que a resposta foi corrigida;
- o total de CPU consumido na execução do programa (soma dos tempos de CPU nas execuções dos casos de teste);
- o total de RAM (em KB) utilizado para executar o código (soma dos consumos de RAM nas execuções dos casos de teste);
- o total do tempo gasto para executar o código (soma dos tempos gastos nas execuções dos casos de teste); e
- a pontuação daquela correção (a mesma é calculada a partir da soma da pontuação de todos os casos de teste cadastrados no desafio pela quantidade dos mesmos).

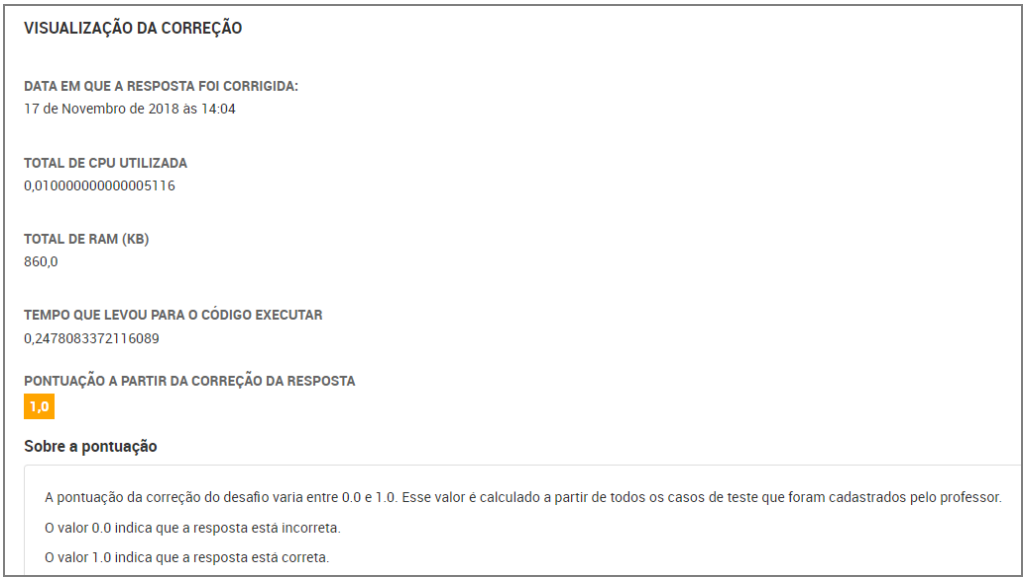
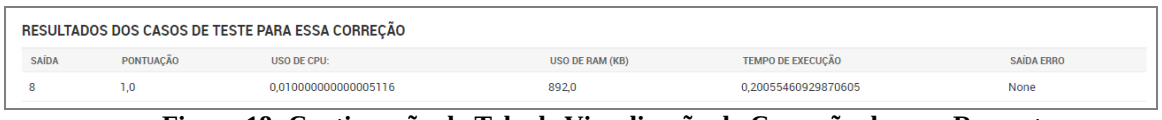


Figura 17: Tela de Visualização da Correção de uma Resposta

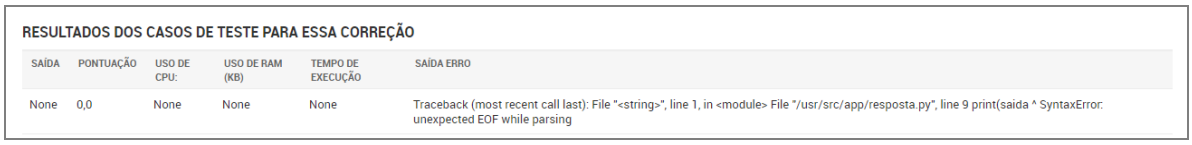
Por fim, a tela apresenta uma tabela com as informações sobre os resultados individuais dos casos de teste, contendo, para cada um (Figura 18): saída; pontuação; o total de CPU; o total de RAM (em KB); o tempo de execução; e a saída de erro.


RESULTADOS DOS CASOS DE TESTE PARA ESSA CORREÇÃO

SAÍDA	PONTUAÇÃO	USO DE CPU:	USO DE RAM (KB)	TEMPO DE EXECUÇÃO	SAÍDA ERRO
8	1,0	0,010000000000005116	892,0	0,20055460929870605	None

Figura 18: Continuação da Tela de Visualização da Correção de uma Resposta

A saída de erro é apresentada se o código-fonte da resposta em questão executar com erro. A Figura 19 ilustra a correção de um código-fonte que possui um erro de sintaxe.


RESULTADOS DOS CASOS DE TESTE PARA ESSA CORREÇÃO

SAÍDA	PONTUAÇÃO	USO DE CPU:	USO DE RAM (KB)	TEMPO DE EXECUÇÃO	SAÍDA ERRO
None	0,0	None	None	None	Traceback (most recent call last): File "<string>", line 1, in <module> File "/usr/src/app/resposta.py", line 9 print(saida ^ SyntaxError: unexpected EOF while parsing

Figura 19: Continuação da Tela de Visualização da Correção de uma Resposta

É importante levar em consideração que essas informações de execução do código-fonte Python (tempo de CPU, total de RAM e tempo de execução) levam em consideração o ambiente virtualizado no qual o código-fonte é executado. O *Docker* fornece a opção de configurar o hardware da máquina virtual, o que permite maior controle sobre o ambiente de execução no servidor.

5. Considerações Finais

O presente trabalho apresentou os detalhes da reformulação e implementação do ambiente CodeLive, incluindo suas características: um software voltado para a execução de código *Python* em um ambiente virtualizado e protegido.

Com base nisso, este necessitou ser pensado de maneira a definir uma arquitetura de software que permitisse a execução isolada de código *Python* (sem interferir no ambiente de execução do software), bem como a utilização de conceitos e ferramentas de virtualização para sua implementação.

Os resultados obtidos no software permitem ao Aluno executar código *Python* de forma on-line no ambiente (com a utilização do *Docker*) e, desta forma, realizar a submissão da resposta a um desafio, sendo este um problema de programação formulado e cadastrado por um Professor.

É importante notar que a característica da execução desse software (utilizar *Docker* e virtualização) requer do servidor uma maior disponibilização de recursos. A máquina utilizada para a execução do ambiente tem como características: frequência de 1.70GHz, 2 núcleos da CPU e 4 GB de memória RAM. Essa foi uma das limitações encontradas durante o desenvolvimento do software, uma vez que recurso computacional utilizado não era equivalente ao de um servidor.

Por fim, o CodeLive, que tem como objetivo proporcionar um contato mais prático entre aluno e professor, foi estruturado para facilitar a interação entre ambos, bem como fornecer recursos para o Professor criar desafios para os Alunos, corrigir respostas dos alunos automaticamente (sem precisar revisar código-fonte) e, para os Alunos, permitir maior compreensão da execução do código-fonte *Python* de forma online, sem a necessidade de configurar um ambiente local de desenvolvimento.

Referências

CARDOSO, D. C. et al. CODE LIVE: Gamificação de um ambiente virtual de programação. ENCOINFO - Congresso de Computação e Sistemas de Informação, Palmas, aug. 2016. Disponível em: <http://ulbra-to.br/encoinfo/encoinfo2016/paper/view/348>. Acesso em: 29 Aug. 2017.

DIEDRICH, Cristiano. O que é Docker? 2015. Disponível em: <https://www.mundodocker.com.br/o-que-e-docker/>. Acesso em: 17 abr. 2019.

ISSN: 2447-0767

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

GOMES, Rafael; SOUZA, Rodrigo. Docker - Infraestrutura como código, com autonomia e replicabilidade. Superintendência de Tecnologia da Informação - Universidade Federal da Bahia (ufb), Salvador, Ba, p.1-4, ago. 2015.