UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO CORREIA GONZALEZ

UM ESTUDO SOBRE O USO DO MECANISMO DE DICAS NO ENSINO DE CONCEITOS BÁSICOS DE PROGRAMAÇÃO

MONOGRAFIA

CAMPO MOURÃO

2017

GUSTAVO CORREIA GONZALEZ

UM ESTUDO SOBRE O USO DO MECANISMO DE DICAS NO ENSINO DE CONCEITOS BÁSICOS DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso de graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva

Coorientador: Prof. Dr. Igor Scaliante Wiese

CAMPO MOURÃO

2017

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **19:30** do dia **23 de novembro de 2017** foi realizada na sala **E103** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Gustavo Correia Gonzalez** com o título **Um Estudo Sobre o Uso do Mecanismo de Dicas no Ensino de Conceitos Básicos de Programação**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Marco Aurélio Graciotto Silva** (orientador), **Prof. Dr. Igor Scaliante Wiese**, **Prof. Dr. Rafael Liberato Roberto** e **Prof. Dr. Marcos Silvano Orita Almeida**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso **2** e atribuiu, em consenso, a nota _____ (_____). Este resultado foi comunicado ao(à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações: _____

_____

_____

_____

_____

_____

Campo Mourão, **23 de novembro de 2017**

Prof. Dr. Igor Scaliante Wiese Membro 1	**Prof. Dr. Rafael Liberato Roberto** Membro 2
Prof. Dr. Marcos Silvano Orita Almeida Membro 3	**Prof. Dr. Marco Aurélio Graciotto Silva** Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Agradecimentos

Agradeço primeiramente ao meu orientador, o Prof. Dr. Marco Aurélio, pela orientação, apoio e empenho. Agradecimentos também ao coorientador, Prof. Dr. Igor Wiese, pelas reuniões iniciais para a elaboração deste trabalho e sua orientação no TCC1, junto com o Prof. Dr. Marco Aurélio que era o coorientador.

Meus agradecimentos aos amigos João Martins, Felipe Fronchetti e Vinicius Morais, companheiros de trabalhos e de grandes debates para resolver as tarefas da universidade, sendo irmãos na amizade que fizeram parte da minha formação.

Agradeço aos meus pais Estaban e Angela, pelo amor, carinho e paciência nesses anos que estive fora de casa. Por não medirem esforços para que eu pudesse levar meus estudos adiante e pela confiança que tiveram ao permitir cursar uma universidade tão distante. Agradecimentos ao meu irmão Guilherme, que sempre me apoiou e divertiu em momentos difíceis.

Por fim, meus eternos agradecimentos a Isabele, minha namorada. Por estar sempre ao meu lado, me apoiar e animar em todas as dificuldades desses anos. Agradeço a Deus por ter colocado no meu caminho uma mulher tão especial que amo muito e por dar forças para enfrentar a graduação.

Resumo

Gonzalez, Gustavo. Um Estudo Sobre o Uso do Mecanismo de Dicas no Ensino de Conceitos Básicos de Programação. 2017. 56. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2017.

Contexto: O elevado nível de reprovação em disciplinas em que são ensinados conceitos básicos de programação, em qualquer grau de ensino, é um problema enfrentado por muitos alunos e tem sido alvo de várias pesquisas. Existe um conjunto de razões que estão relacionadas com a origem do problema, como o método de ensino e aprendizagem, a falta de algumas competências e interesse por parte dos alunos, e a própria dificuldade do tema. Para tratar esse problema, sistemas que auxiliam o aprendizado com dicas vêm sendo utilizados para ajudar os alunos na realização de exercícios.
Objetivo: O objetivo deste trabalho foi desenvolver um sistema de dicas de código aberto para auxiliar os alunos na aprendizagem de conceitos básicos de programação, mais especificamente nos tópicos de estrutura de condição e laço de repetição.
Método: O sistema de dicas foi desenvolvido com o *framework* de desenvolvimento Laravel. Ele foi avaliado em um estudo com alunos e profissionais da área da Computação a partir de um conjunto de exercícios previamente definidos. O estudo foi dividido em 3 etapas, sendo às duas primeiras destinadas ao preenchimento da base de dicas do sistema. Nessas etapas, participaram alunos que já realizaram as matérias de algoritmos e estruturas de dados e profissionais da área de Computação. Na terceira etapa do estudo, alunos do primeiro período utilizaram o sistema para resolução de exercícios com auxílio das dicas previamente cadastradas. Após a realização destas etapas, por meio dos dados capturados nas submissões, foram extraídas as medidas referentes aos códigos e utilização das dicas para resolução dos exercícios. Os alunos da terceira etapa também responderam um questionário sobre o uso da ferramenta.
Resultados: Foram criadas 79 dicas e realizadas 243 submissões no estudo. Considerando os participantes da terceira etapa do estudo, observam-se indícios de que as dicas foram úteis à resolução dos primeiros exercícios. Os resultados do questionário indicam que a experiência da utilização do sistema de dicas foi positiva e que as dicas foram úteis para o desenvolvimento dos exercícios. No entanto, a quantidade de participantes e exercícios resolvidos para a última etapa do estudo não permitem uma avaliação mais rigorosa dos resultados e da ferramenta.

Conclusões: Neste trabalho, foi desenvolvido um sistema de dicas, relatando-se suas principais características e avaliando-o com alunos e profissionais da área de Computação. A partir da avaliação realizada após a aplicação do estudo, foi possível identificar indícios de que a utilização do mecanismo de dicas ajudaram os alunos na resolução de exercício que abordam conceitos básicos de programação.

Palavras-chaves: Educação, Computação, Programação, Dicas

Abstract

Gonzalez, Gustavo. A Study on the Use of the Mechanism of Hints for Teaching Basic Programming Concepts. 2017. 56. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2017.

Context: The high level of reprobation in subjects where basic programming concepts are taught at any level of education is a problem faced by many students and has been the subject of a number of researches. There are a number of reasons that are related to the origin of the problem, such as the teaching and learning method, the lack of some skills and interest on the part of the students, and the difficulty of the subject itself. To address this problem, systems that aid learning with hints are being used to help students perform exercises.
Objective: The objective of this work was to develop an open source hints system to help students learn basic programming concepts, more specifically on condition structure and loop repetition topics.
Method: The hints system, called iHint, was developed with the Laravel development framework. It was evaluated in a study with students and professionals of the Computing area from a set of previously defined exercises. The study was divided in 3 stages, being the first two destined to fill the base of hints of the system. In these stages, students who already did the subjects of algorithms and data structures and professionals of the Computing area participated. In the third stage of the study, students of the first period used the system to solve exercises using the previously registered hints. After completing these steps, through the data captured in the submissions, the measures related to the codes and the use of the hints for resolving the exercises were extracted. The third stage students also answered a questionnaire about the use of the tool.
Results: 79 hints were created and 243 submissions were made in the study. Considering the participants of the third stage of the study, there are indications that the hints were useful to solve the first exercises. The results of the questionnaire indicate that the experience of using the system of hints was positive and that the hints were useful for the development of the exercises. However, the number of participants and exercises solved for the last stage of the study does not allow a more rigorous evaluation of the results and the tool.
Conclusions: In this work, a system of hints was developed, reporting its main characteristics and evaluating it with students and professionals of the Computing area. From the evaluation performed after the application of the study, it was possible to identify indications that the use

of the mechanism of hints helped the students in the resolution of exercise that approach basic concepts of programming.

Keywords: Education, Computing, Hints, Learnsourcings

Lista de figuras

Lista de tabelas

Sumário

Capítulo 1

Introdução

O ensino de linguagens de programação visa propiciar aos alunos o desenvolvimento de um conjunto de competências necessárias para conceber programas e sistemas computacionais capazes de resolver problemas reais. A reprovação dos estudantes em disciplinas de programação é um tema que tem sido alvo de alguns estudos (BOSSE; GEROSA, 2015; CUKIERMAN, 2015).

O problema do insucesso é evidenciado por Lahtinen et al. (2005), que realizaram uma pesquisa com diferentes universidades para estudar as dificuldades na aprendizagem de programação. Como resultado, foi percebido que as questões mais difíceis na programação são: a compreensão de como projetar um programa para resolver uma tarefa determinada; dividir as funcionalidades em procedimentos; e encontrar erros de seus próprios programas.

Estas dificuldades contribuem para o baixo rendimento dos alunos nas disciplinas de programação. Por isso os pesquisadores estão utilizando vários métodos e softwares para minimizar esse problema. Portanto, a utilização de sistemas de dicas, tutores inteligentes e ferramentas de apoio aos alunos estão sendo aplicados em várias escolas e universidades.

O presente trabalho foca em sistemas de dicas, no qual é reproduzido o contato que o aluno teria com uma pessoa mais instruída quando precisa tirar uma dúvida ou pedir um auxílio para realizar um exercício.

Observa-se, na literatura, que a maioria das pesquisas apresenta soluções que se baseiam na utilização de inteligência artificial ou a interferência dos professores na criação das dicas (ELKHERJ; FREUND, 2014; RAZZAQ; HEFFERNAN, 2010; PRICE, 2015; ANTONUCCI et al., 2015; PAQUETTE et al., 2012). Por exemplo, com a necessidade de prover um suporte personalizado aos alunos, Elkherj e Freund (2014) criaram um sistema de dicas que permite ao professor enviar uma dica personalizada após verificar que o aluno realizou várias tentativas para resolver um exercício. Entretanto, com o aumento do número de alunos utilizando o sistema, os professores não conseguem oferecer suporte a todos eles. Assim, utilização dessa abordagem na construção do sistema de dicas apresenta limitações em relação à quantidade de alunos realizando os exercícios.

O objetivo desse trabalho foi criar um software de código aberto para auxiliar na aprendizagem de conceitos básicos de programação, implementando um sistema colaborativo de dicas escritas pelos próprios usuários.

Este estudo aplicou o mecanismo de dicas para que apenas os alunos estejam envolvidos na criação das dicas, assim possibilitando maior aprendizado do conteúdo dos exercícios. O método aplicado para a implementação do sistema é o *learnersourcing* apresentado por Glassman et al. (2016), que gerencia as atividades dos alunos por meio de uma *interface* que coleta os dados de aprendizagem dos usuários, suas avaliações de explicações e elicita a geração de novas respostas para futuros alunos. Desta forma, os alunos podem, mediante de sua própria experiência resolvendo exercícios, criar dicas úteis de acordo com implementações, gerando sugestões para colegas com base em seu conhecimento.

Para avaliar o sistema de dicas, foi realizado um estudo organizado em 3 etapas, aplicando-se 7 exercícios de programação na linguagem C abordando estrutura de condição e laço de repetição. Sendo assim, a primeira etapa foi realizada com alunos experientes da graduação do curso de Bacharelado em Ciência da Computação (BCC) da Universidade Tecnológica Federal do Paraná no campus Campo Mourão (UTFPR-CM), visando o preenchimento da base de dicas do sistema. A segunda etapa também teve o objetivo de preencher a base de dicas, mas foi realizado com profissionais da área da Computação. Posteriormente, a terceira etapa foi realizada com os alunos do primeiro semestre do curso de BCC da UTFPR-CM, que utilizaram o sistema com o auxílio das dicas criadas nos estudos anteriores. Nessa etapa, os alunos também realizaram a criação de dicas conforme a primeira e segunda etapa do estudo. Para realizar a avaliação da ferramenta, foram utilizados dados referentes às submissões dos alunos da terceira etapa e questionário respondido por esses alunos.

Os próximos capítulos estão organizados da seguinte forma. O Capítulo 2 apresenta o referencial teórico e os trabalhos relacionados. O Capítulo 3 apresenta o sistema de dicas desenvolvido neste trabalho. O Capítulo 4 apresenta o estudo executado para avaliar a abordagem proposta, os resultados obtidos e respectiva discussão. Por fim, o Capítulo 5 apresenta a conclusão deste trabalho e trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo apresentamos uma revisão da literatura de modo a fundamentar a construção deste trabalho. A Seção 2.1 trata sobre os sistemas de dicas aplicados em áreas distintas do conhecimento. Por fim, na Seção 2.2 são apresentados os trabalhos sobre a aplicação do sistema de dicas para o ensino de programação.

2.1. Sistemas de dicas para áreas distintas de Programação

Esta seção aborda sobre sistema de dicas que estão sendo desenvolvidos em outros âmbitos educacionais, nos quais os pesquisadores estão utilizando recursos computacionais para melhorar o aprendizado de alunos.

Elkherj e Freund (2014) apontam que as sugestões que são apresentadas pelo sistema de aprendizagem sem o aluno requisitar e antes que realize as tentativas, não dependendo das tentativas mal sucedidas que o estudante tenha realizado, limitam severamente a eficácia das sugestões.

Desta forma, eles desenvolveram um sistema alternativo para dar dicas aos estudantes. A principal diferença é que, ao invés de apenas apresentar as dicas para os alunos, o sistema permite que um instrutor envie uma dica para um estudante após este ter feito várias tentativas falhas para resolver o problema. Depois de analisar os erros do aluno, o instrutor é mais capaz de entender o problema no pensamento do aluno e enviar uma dica mais útil. O sistema foi implantado em um curso de probabilidade e estatística com 176 alunos, obtendo *feedback* muito positivo dos alunos. No entanto, um desafio que os autores enfrentaram foi como escalar efetivamente o sistema de forma que todos os alunos que precisam de ajuda obtenham dicas eficazes. Atualmente, com uma grande quantidade de alunos, ficaria exaustivo para os instrutores avaliarem cada submissão dos exercícios de cada aluno para retornarem uma dica específica do erro cometido. Então eles criaram

um banco de dados de dica que permite que os instrutores reutilizem, compartilhem e melhorem a partir de sugestões escritas anteriormente.

Cummins et al. (2016) investigaram o uso de dicas para 4,652 usuários qualificados em um ambiente de aprendizagem *on-line* de grande escala chamado Isaac, que permite aos usuários responder a perguntas de Física com até cinco dicas. Foi investigado o comportamento do usuário ao usar dicas, engajamento dos usuários com desvanecimento (o processo de tornar-se gradualmente menos dependentes das dicas fornecidas) e estratégias de dicas incluindo decomposição, correção, verificação ou comparação. Como resultados obtidos, os alunos apresentaram estratégias para as resoluções dos exercícios, sendo a mais comum é ver o conceito da dica para realizar a decomposição do problema e, em seguida, enviar uma resposta correta. A outra estratégia é usar os conceitos da dica para determinar se a pergunta pode ser respondida. No entanto, uma grande proporção dos usuários que utilizaram essa estratégia acabou não tentando responder a pergunta.

Razzaq e Heffernan (2010) realizaram uma pesquisa para avaliar a melhor forma de apresentar as dicas para os estudantes. Existem muitos sistemas que permitem aos alunos pedir sugestões quando precisam de assistência para resolver problemas, mas muitos alunos apresentam dificuldades em saber quando pedir uma dica. Então os pesquisadores avaliaram se é melhor dar uma dica quando um aluno comete um erro ou esperar até que o aluno peça uma dica. Foi utilizado o *Assessment System* que é usado principalmente por professores do ensino médio em *Massachusetts* que estão preparando estudantes para o teste do Sistema de Avaliação Integral de Massachusetts (MCAS). Esse sistema é utilizado em aulas regulares de matemática e para trabalhos de casa.

Cada dica oferecida pelo *Assessment System* mostra uma mensagem com as informações e sugestões para resolver um problema específico e faz parte de uma sequência de sugestões de 3 a 5 dicas. Para cada sequência de sugestões, a última dica indica ao aluno exatamente o que fazer ou dá a resposta correta. O estudo foi dividido em dois tipos: sugestões sob demanda e dicas proativas. As sugestões sob demanda apresentaram aos alunos uma dica apenas quando clicaram no botão de dica e as dicas proativas apresentaram aos alunos uma dica sempre que cometiam um erro. Com isso, os pesquisadores aplicaram o estudo em uma turma da oitava série com 72 alunos, cujos alunos receberam quatro problemas matemáticos e o sistema forneceu sugestões sob demanda ou proativas de forma aleatória para os alunos.

Foi descoberto que os estudantes melhoraram significativamente com sugestões sob demanda em relação à quando o sistema mostrava uma dica quando ocorreu um erro. Os alunos que tendem a pedir uma grande quantidade de dicas aprendem significativamente mais com sugestões sob demanda, mas para estudantes que pediram um número baixo de dicas, não houve diferença significativa entre as duas condições.

Glassman et al. (2016) criaram um sistema de dicas chamado *Dear Beta* para auxiliar na criação de projeto de circuitos digitais que exigem mais experiência dos alunos. Aplicando o método do *learnersourcing*, os alunos apresentaram mais motivação para se envolver no conteúdo

de aprendizagem, beneficiando-se pedagogicamente da própria tarefa de realizar exercícios e produzir dicas. No estudo realizado pelos pesquisadores, 9 dos 226 alunos do curso de arquitetura de computadores participaram do estudo. Estes alunos foram recrutados através de um fórum. Esse estudo foi realizado para estudar a eficácia das dicas para otimizar os circuitos para que usassem menos transistores.

Após algumas semanas de estudo, o número de voluntários aumentou e chegou a 20 alunos, mas na última semana do estudo o número de usuários registrados no sistema *Dear Beta* aumentou linearmente de 20 para 166. Nos 9 dias entre o lançamento do *Dear Beta* e a data de vencimento do estudo de laboratório, os usuários adicionaram 76 erros de verificação e 57 sugestões como uma resposta a esses erros. Metade dos erros recebeu pelo menos uma dica. Os resultados do estudo de implantação e subsequente estudo de laboratório demonstram a viabilidade desses fluxos de trabalho e indicam que as dicas geradas pelo aluno são úteis para os alunos.

2.2. Sistemas de dicas para o ensino de Programação

Esta seção mostra as pesquisas focadas no ensino de programação. Os pesquisadores estão aplicando diversas abordagens, na maioria dos casos, com alunos iniciantes de disciplinas de Programação.

Price (2015) utilizou um sistema de tutores inteligentes (STI) para auxiliar os alunos na ausência de instrutores, fornecendo sugestões e advertências para os alunos que precisam de ajuda. Além disso, as técnicas baseadas em dados podem gerar *feedback* automaticamente a partir de tentativas de resolução de um problema. Esse *feedback* é criado através da base de dados de submissões realizadas pelos alunos durante a utilização do sistema.

Antonucci et al. (2015) desenvolveram ferramentas e técnicas que suportam automaticamente estudantes na resolução de exercícios de programação, visando fomentar a adoção de cursos *on-line* e MOOCs para o aprendizado de programação. Eles desenvolveram e avaliaram um sistema de dica incremental para exercícios de programação. Este sistema reage ao pedido do aluno pelas dicas, apresentando uma série de sugestões sobre como abordar e solucionar o problema. Essas sugestões são criadas antecipadamente a partir do código fonte da solução correta, podendo inclusive revelar partes do código da solução. Também existe um modo manual, em que o professor pode personalizar a dica.

O sistema foi avaliado durante o curso de introdução a programação, que oferece um módulo *on-line* para complementar o aprendizado dos estudantes. Para isso foi utilizado o Moodle integrado com o Codeboard 2, que permite aos alunos escrever, compilar, testar e enviar programas a partir do navegador. Desse modo, 38 alunos utilizaram o sistema de dicas e, com a análise realizada pelos pesquisadores, foi observado que os alunos que não usaram as dicas apresentaram soluções corretas quase sempre (99,5%). Quanto aos alunos que utilizaram as dicas, apenas 63% criaram soluções corretas. Uma possível explicação foi que os estudantes que utilizaram dicas e criaram

soluções incorretas (37%) precisariam de mais assistência. Uma maneira de alcançar isso seria a adição de mais níveis de sugestão e explicações dentro das dicas Antonucci et al. (2015). Os estudantes também preencheram um questionário sobre a utilização dos sistemas de dicas, sendo que 16 alunos (42%) utilizaram o sistema de dica em pelo menos um exercício. Entre eles, 5 estudantes (33%) descobriram que o sistema de dica era muito difícil de usar, enquanto que para o restante deles (66%) afirmou que era fácil de usar. Antonucci et al. (2015) constataram que, de modo geral, 73% dos alunos que utilizaram o sistema de dicas acharam útil e 80% afirmaram que o nível de granularidade das dicas eram apropriados. Foi concluído que o sistema foi capaz de processar todos os exercícios submetidos pelos alunos, produzindo sugestões satisfatórias. Como também sempre que necessário, os professores poderiam facilmente redefinir a política de processamento, definindo uma tabela de dicas personalizada. Os dados coletados usando questionários e dados de uso sugerem que o sistema de dica foi uma adição benéfica à plataforma de programação de ensino na nuvem.

Paquette et al. (2012) desenvolveram o framework ASTUS, que modela a representação do conhecimento do ponto de vista de professores a partir das tarefas de programação. O modelo é gerado usando conhecimento semântico necessário para a realização de exercícios. Cada conceito que compõe este conhecimento define um conjunto de características essenciais que podem se referir a outros conceitos ou valores primitivos (inteiro, número, decimal, símbolo, booleano). Para cada tarefa, o ASTUS cria um modelo único, definindo as estruturas que o aluno precisa manipular. Assim, é possível gerar automaticamente dicas para auxiliar o estudante para o próximo passo da realização da tarefa.

O sistema oferece dicas a partir do avanço do aluno na realização da tarefa. A cada passo atingido da tarefa, o sistema apresenta uma dica que referencia o passo seguinte a ser realizado. As dicas oferecidas são divididas entre: dica sobre a estrutura (independente da tarefa) e sobre o conteúdo (específico para a tarefa). Foram definidas as estruturas das dicas como modelos de texto a serem preenchidos com o conteúdo específico da tarefa extraído das unidades de conhecimento definidas no modelo da tarefa.

Para avaliar o ASTUS, Paquette et al. (2012) aplicaram estudos com dois grupos de alunos. O primeiro grupo teve a participação de professores que auxiliaram os alunos quando necessário, enquanto o segundo grupo obteve o acesso ao tutor projetado usando o ASTUS. Os resultados mostraram que as sugestões geradas pelo ASUTS podem ser tão eficientes quanto as de autoria dos professores.

2.3. Considerações Finais

Muitos pesquisadores estão estudando e aplicando diferentes estratégias para melhorar o ensino e desenvolver melhor as habilidades dos alunos em resolver problemas. Este estudo adota como base para a criação do sistema de dicas o estudo realizado por Glassman et al. (2016), que desenvolveu um

sistema de dicas para auxiliar projetos de circuitos digitais no curso de arquitetura de computadores. Deste modo, foi construído um sistema que difere do apresentado por Elkherj e Freund (2014), em que o professor apresenta uma dica em tempo real a um aluno que tenha realizado várias tentativas para resolver um exercício. Esse tipo de sistema não é escalável para uma grande quantidade de usuários. O objetivo é criar um sistema que não seja necessário que um professor crie dicas para os alunos. Assim, os próprios alunos criam dicas para ajudar outros colegas e, concomitantemente, melhoram seu desempenho em programação.

Capítulo 3

Sistema de Dicas

Este capítulo descreve todas as etapas para o desenvolvimento do sistema de dicas feito neste trabalho, disponibilizado como aplicação Web[1], e mantido no *GitHub*[2] sob a licença *MIT*. A Seção 3.1 apresenta as tecnologias utilizadas na implementação e gestão do sistema e a Seção 3.2 explica as funcionalidades que o sistema possui.

3.1. Tecnologias

Esta seção apresenta as tecnologias utilizadas neste trabalho. A Seção 3.1.1 descreve as características da linguagem de programação utilizada. A Seção 3.1.3 mostra uma definição e as característica de um *framework*. A Seção 3.1.2 apresenta a arquitetura MVC, utilizada pela maioria dos atuais *frameworks web*. A Seção 3.1.4 apresenta o *framework* utilizado no desenvolvimento do sistema de dicas. A Seção 3.1.5 refere-se ao padrão de projeto *Repository Design Pattern*, acrescentado ao MVC. A Seção 3.1.6 mostra o gerenciador de dependências utilizado. Por fim, a Seção 3.1.7 mostra como se realizou a extração das medidas dos códigos submetidos.

3.1.1. Linguagem PHP

O *Personal Home Page* (PHP) é uma linguagem interpretada, geralmente usada para o desenvolvimento de aplicações no lado do servidor, capaz de gerar conteúdo dinâmico na Web (PHPGROUP, 2017b). O código é interpretado no lado do servidor pelo módulo PHP, que também gera a página *web* a ser visualizada no lado do cliente. O PHP tem passado por uma grande evolução nos últimos anos, consolidando-se entre as linguagens mais importantes para o desenvolvimento *web*, sendo utilizado por grandes empresas e projetos de código aberto.

[1] A aplicação está disponível em <http://138.197.124.212/>.
[2] <https://github.com/gustavoCorreiaGonzalez/iHint>

Atualmente, ele conta com uma sólida base de orientação a objetos, funcionalidades como *closures*, *generators* e *traits*, iniciativas de padronização de código com o PHP-FIG e as PSRs, ferramentas como *Composer* para o gerenciamento de dependências, um vasto e confiável repositório de bibliotecas como o *Packagist* e uma comunidade bastante ativa.

Uma das mudanças mais importante para o amadurecimento do PHP foi a sólida implementação do paradigma orientado a objetos. O PHP nasceu procedural e ao longo do tempo foi se tornando orientado a objetos. Uma das maiores vantagens da programação orientada a objetos é permitir uma adequada representação de um problema, com melhor organização e alto reaproveitamento de código. Conceitos como classes, métodos, propriedades, herança, *namespaces*, instâncias entre outros fazem parte do vocabulário de uma linguagem orientada a objetos (PHPGROUP, 2017a).

3.1.2. MVC e Arquitetura

O *Model-view-controller* (MVC) é um padrão de arquitetura de *software* que permite organizar o projeto em componentes, separando a lógica de negócio da lógica de apresentação. O fluxo de funcionamento do MVC, apresentado na Figura 3.1, é simples: o *controller* recebe as requisições, solicita as informações necessárias para o *model* e envia as informações para a *view* organizar e apresentar o conteúdo.

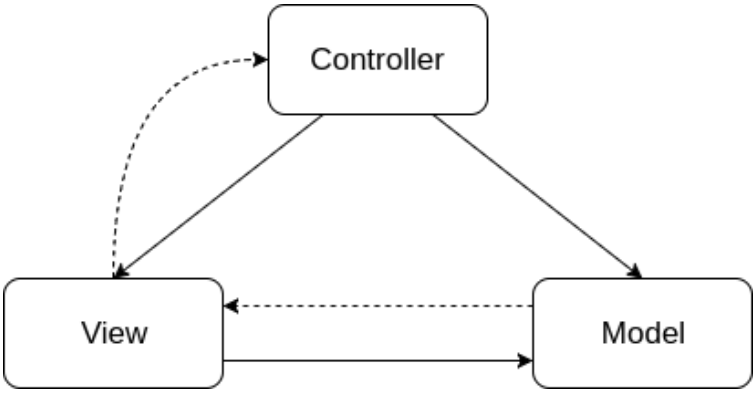

Figura 3.1. Visão geral do sistema de dicas.
Fonte: Reenskaug (1979).

Com esta organização em componentes, as responsabilidades ficam melhor separadas. O *controller* apenas recebe a requisição e retorna um resultado. Ele não sabe se comunicar com o banco de dados e isso não é de sua responsabilidade. Ele apenas repassa esta tarefa para o seu responsável. O *model* é responsável pela lógica do negócio e pela comunicação com a base de dados. Após realizar as suas tarefas, o *model* retorna os dados para o *controller*, que não sabe sobre como apresentar o conteúdo e, por sua vez, retorna estes dados para a *view*. A *view* é responsável pela apresentação do conteúdo.

3.1.3. *Framework*

Frameworks são descritos por Bosch et al. (2000) como um conjunto de classes que incorpora um design abstrato para soluções para uma família de problemas relacionados. Também oferece um conjunto de funcionalidades, realizando responsabilidades para subsistemas típicos do domínio das aplicações (FAYAD; SCHMIDT, 1997), tais como acesso a banco de dados, gerenciamento de sessões, validação de dados, geração de código, internacionalização da aplicação, autenticação, dentre outros.

Um *framework* web é utilizado para solucionar um tipo específico de problema que é a criação de aplicações web. Costuma ter uma arquitetura e organização pré-definida decorrente de decisões de projeto amplamente discutidas e da experiência de seus criadores e colaboradores. Por exemplo, *frameworks* web geralmente implementar o padrão arquitetural MVC, dada sua adequação ao domínio de aplicações Web.

Um *framework web* é visto como um esqueleto de uma aplicação. Ele possui partes fixas que são implementadas pelo *framework* e partes variáveis que são implementadas pelo programador para especificar as funcionalidades de sua aplicação. Existem diversos *frameworks* web em PHP, como o Zend *Framework*[3], Symfony[4] e Laravel[5], sendo que este último foi escolhido para utilização neste trabalho.

3.1.4. *Framework* Laravel

Para o desenvolvimento do sistema de dicas, foi escolhido o *framework* Laravel por causa do conhecimento prévio da linguagem PHP e do *framework*. A arquitetura utilizada é a MVC, que auxilia o desenvolvedor a criar aplicações seguras e performáticas de forma rápida, com código limpo e simples, inclusive utilizando o padrão PSR-2 como guia para estilo de escrita de código. O *Framework* Laravel é feito em PHP e atualmente está em sua versão 5.5.

Para a criação de *interface* gráfica, o Laravel utiliza uma *Engine* de *template* chamada Blade, que fornece ferramentas que auxiliam na construção de *interfaces* funcionais de forma rápida e ajudam a evitar a duplicação de código.

A comunicação com o banco de dados é realizada através de biblioteca *Eloquent ORM*, que permite consultar e manipular dados de um banco de dados usando um paradigma orientado a objetos. Portanto, o *Eloquent ORM* encapsula o código necessário para manipular os dados, de modo que não se utiliza a linguagem SQL, mas a linguagem de programação que está utilizando para desenvolver o sistema.

[3] <https://framework.zend.com/>
[4] <https://symfony.com/>
[5] <https://laravel.com/>

3.1.5. *Repository Design Pattern*

Além de utilizar o padrão de arquitetura MVC implementado pelo Laravel, também se teve a necessidade de adicionar um padrão de projeto para complementar a arquitetura, visando a organização e reaproveitamento de código. Portanto, foi adicionado o *Repository Design Pattern*, que compõe a camada de serviços em que se concentram as regras de negócio. É nessa camada que ocorre o acesso aos dados e entidades do sistema que se localizam na camada *models*. Os repositórios permitem que o acesso e a interação com os *models* não sejam feitos diretamente em outras partes do sistema, conforme apresentado na Figura 3.2.

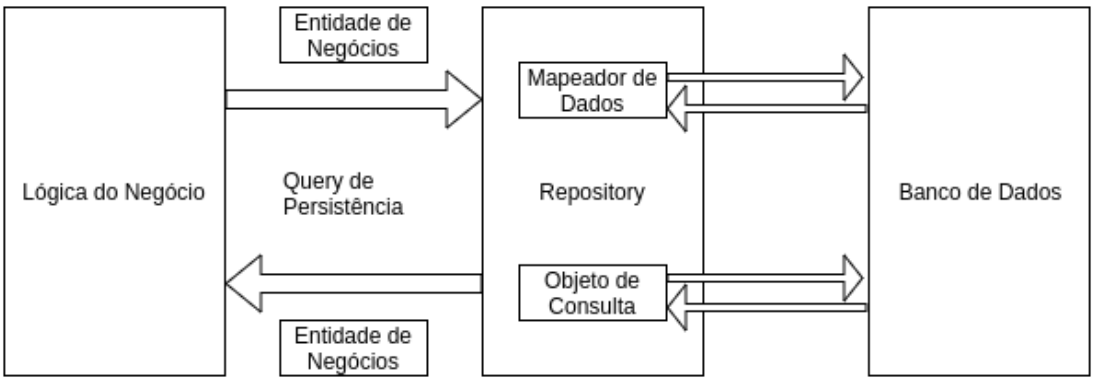

Figura 3.2. Repository Pattern.
Fonte: Evans (2004).

Com a utilização dos repositórios, foi possível alcançar as seguintes característica no desenvolvimento da aplicação:

- Centralizar regras de recuperação e persistência de dados.
- Abstrair a utilização de ORMs, possibilitando a troca por outros ORMs
- Abstrair o tipo de banco de dados, que, assim como ORMs, pode ser trocado.
- Devolver dados em um formato agnóstico (*array* ou *StdClass*), sem acoplamento ao ORM.

Apesar da utilização do *Repository Design Pattern* não ser essencial para o funcionamento da aplicação desenvolvida neste trabalho, o padrão de projeto em questão foi escolhido por oferecer um código mais limpo e legível, facilitando a manutenção. Ele também possibilitou a organização das regras de negócio que,de outra forma, ficariam espalhadas nos *models*.

3.1.6. Gerenciador de Dependências

As dependências são os pacotes ou bibliotecas de que o projeto depende para o seu funcionamento. A utilização de pacotes permite que o desenvolvedor se preocupe somente com a lógica de negócio da sua aplicação, deixando tarefas como validação de formulários, acesso a banco de dados ou envio de *e-mail* para bibliotecas terceiras. O *Composer* é uma ferramenta para o gerenciamento de

dependências em PHP. Ele permite declarar as dependências do seu projeto e fazer o gerenciamento (instalação e atualização) delas de forma simples e automatizada. Além disto, o Composer possui acesso a um vasto repositório de pacotes, denominado *Packagist*, onde você pode consultar ou publicar pacotes.

3.1.7. Ferramenta para extração de medidas

Para realizar a extração das medidas dos códigos em C que foram submetidas ao sistema de dicas pelos alunos, foi utilizada a ferramenta de análise *C and C ++ Code Counter* (CCCC) (LITTLEFAIR, 1998). O CCCC processa cada um dos arquivos especificados na linha de comando. Para cada arquivo, examina-se a extensão do nome do arquivo e, se a extensão for reconhecida como uma linguagem suportada, o analisador é executado no arquivo. À medida que cada arquivo é analisado, o reconhecimento de certas construções fará com que os registros sejam escritos em um banco de dados interno. Quando todos os arquivos foram processados, um relatório é gerado no formato HTML.

Por padrão, o relatório HTML principal é gerado para o arquivo cccc.htm em um subdiretório chamado .cccc do diretório de trabalho atual, com relatórios detalhados para cada módulo (ou seja, classe C++ ou Java) identificados pela análise. Além dos relatórios HTML, contendo os dados listados na Tabela 3.1, a execução do CCCC produz relatórios resumidos em formato XML.

Tabela 3.1. Tabelas Geradas pelo CCCC

Nome da Tabela	Descrição
Resumo do projeto	Apresenta valores resumidos de várias medidas sobre o corpo do código-fonte submetido.
Resumo Procedural	Apresenta valores de medidas procedurais somados para cada módulo identificado no código enviado.
Detalhes Procedurais	Apresenta valores das mesmas medidas processuais cobertas no relatório de resumo procedural, mas desta vez dividido em cada módulo nas contribuições de cada função membro do módulo.
Resumo Estrutural	Apresenta contagens de relacionamentos de fan-in e fan-out para cada módulo identificado e uma métrica derivada chamada medida Henry/Kafura/Shepperd, que é calculada como o quadrado do produto das contagens de fan-in e fan-out .
Detalhes Estruturais	Apresenta listas dos módulos que contribuem para as contagens de relacionamento relatadas no resumo estrutural.
Extensões Rejeitadas	Apresenta uma lista de regiões de código que o analisador não conseguiu analisar.

Fonte: Littlefair (1998).

A Tabela 3.2 apresenta as possíveis medidas que o CCCC pode extrair dos algoritmos. Para o desenvolvimento do sistema de dicas, foram utilizadas as medidas LOC e MVG, conforme definidas a seguir:

Tabela 3.2. Medidas Exibidas na Análise

Sigla	**Nome da medida**
LOC	Linhas de Código
MVG	Complexidade Cyclomatic de McCabe
COM	Linha de comentários
L_C,M_C	LOC/COM
FO,FOc,FOvFI,FIc,FIc	Fan-out,Fan-in
HKS,HKSv, HKSc	Medida de Henry-Kafura / Shepperd
NOM	Número de módulos
WMC	Métodos ponderados por classes
REJ	Linhas rejeitadas

Fonte: Littlefair (1998).

- Linhas de Código (LOC): Esta contagem segue o padrão da indústria de contar linhas de código-fonte não em branco, sem comentários. As linhas do pré-processador são tratadas como em branco. As declarações de classe e função são contadas, mas as declarações de dados globais são ignoradas. Pode ocorrer uma dupla contagem de linhas nas definições de classe, pois o algoritmo trata o total sobre um módulo como a soma das linhas pertencentes ao próprio módulo e as linhas pertencentes às funções do membro (as declarações e definições das funções do membro no corpo da classe contribuirá para ambas as contagens).
- Complexidade Ciclomática de McCabe (1976) (MVG): A definição formal da complexidade ciclomática é a contagem de caminhos linearmente independentes através de um grafo de fluxo de controle derivado de um subprograma. Uma aproximação pragmática a esta medida pode ser encontrada contando palavras-chave de linguagem e operadores que introduzem resultados de decisão. Isso pode ser mostrado corretamente na maioria dos casos. No caso de C, a contagem é incrementada para cada um dos seguintes tokens: “if”, “while”, “for”, “switch”, “break”, “&&”, “||”.

Essas medidas foram consideradas pois, para os problemas apresentados no estudo, nos quais são aplicados exercícios que abordam conceitos de estruturas condicionais e laço de repetição, as respostas são construídas utilizando módulos e caminhos criados pelo uso dessas estruturas. A medida MVG está relacionada aos caminhos completos distintos do programa, diretamente relacionado com estruturas condicionais e laços de repetição. A quantidade de linhas está relacionada com a eficácia do aluno na implementação das soluções.

Para realizar a captura dos dados para o sistema de dicas, foi necessário realizar um *parsing* do XML gerado pelo CCCC. Foi preciso utilizar o pacote *XML Document Parser* do Laravel, o qual oferece suporte para leitura do XML, similiar àquele apresentado na Figura 3.3.

```xml
<!-- Detailed report on module anonymous -->
<CCCC_Project>
  <module_summary>
    <lines_of_code value="3" level="0"/>
    <lines_of_code_per_member_function value="******" level="0"/>
    <McCabes_cyclomatic_complexity value="0" level="0"/>
    <McCabes_cyclomatic_complexity_per_member_function value="******" level="0"/>
    <lines_of_code value="0" level="0"/>
    <lines_of_code_per_member_function value="********" level="0"/>
    <lines_of_code_per_line_of_comment value="------" level="0"/>
    <McCabes_cyclomatic_complexity_per_line_of_comment value="------" level="0"/>
    <weighted_methods_per_class_unity value="1" level="0"/>
    <weighted_methods_per_class_visibility value="0" level="0"/>
    <depth_of_inheritance_tree value="0" level="0"/>
    <number_of_children value="0" level="0"/>
    <coupling_between_objects value="0" level="0"/>
    <IF4 value="0" level="0"/>
    <IF4_per_member_function value="********" level="0"/>
    <IF4_visible value="0" level="0"/>
    <IF4_visible_per_member_function value="********" level="0"/>
    <IF4_concrete value="0" level="0"/>
    <IF4_concrete_per_member_function value="********" level="0"/>
  </module_summary>
  <module_detail></module_detail>
  <procedural_detail>
    <member_function>
      <name>main(...)</name>
      <extent>
        <description>definition</description>
        <source_reference file="4_attempt_execice1.c" line="4"/>
      </extent>
      <lines_of_code value="3" level="0"/>
      <McCabes_cyclomatic_complexity value="0" level="0"/>
      <lines_of_comment value="0" level="0"/>
      <lines_of_code_per_line_of_comment value="------" level="0"/>
      <McCabes_cyclomatic_complexity_per_line_of_comment value="------" level="0"/>
    </member_function>
  </procedural_detail>
  <structural_detail>
    <module>
      <name>anonymous</name>
      <client_module></client_module>
      <supplier_module></supplier_module>
    </module>
  </structural_detail>
</CCCC_Project>
```


Figura 3.3. Exemplo de Saída Gerada Pelo CCCC.
Fonte: Littlefair (1998).

3.2. Implementação

Esta seção descreve o sistema de dicas desenvolvido. O caso de uso da Figura 3.4 apresenta as partes que compõem o *software*. A preocupação com a utilização de todas as tecnologias descritas na Seção 3.1 levou ao desenvolvimento de um sistema flexível e de fácil manutenção, cujo funcionamento está baseado em um navegador *web*.

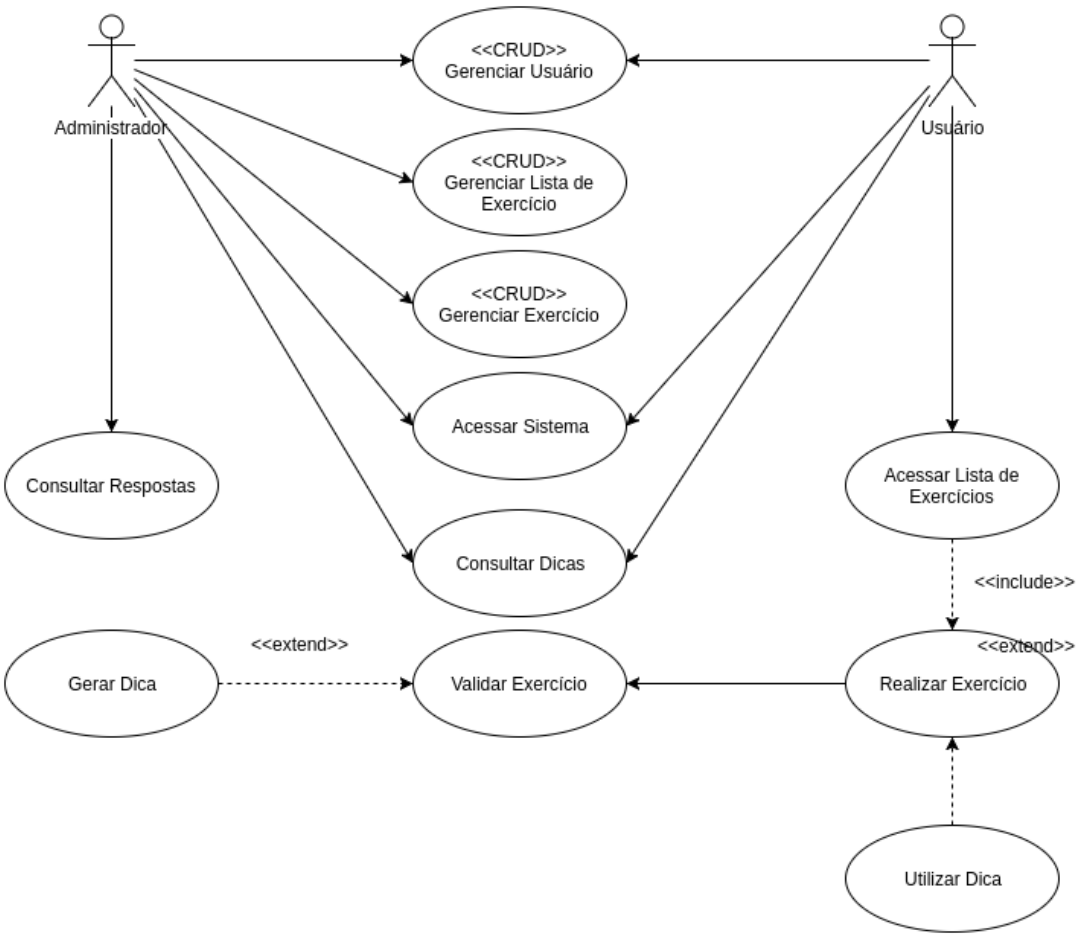

Figura 3.4. Visão geral do sistema de dicas.

O sistema apresenta funcionalidades de acesso e registro, em que o usuário poderá se cadastrar escolhendo a opção de aluno ou administrador. O aluno realiza exercícios disponíveis no banco de dados do sistema, cria dicas para cada exercício e pode consultar dicas de outros usuários. O administrador pode gerenciar usuários, consultar submissões e dicas realizadas pelos alunos e criar listas de exercícios. O usuário pode navegar através dos menus da tela inicial, que contém as opções de cadastro e acesso do sistema de dicas:

Acessar Sistema: módulo responsável pela entrada dos usuários no sistema. É através deste módulo que o usuário tem acesso às funcionalidades habilitadas para seu nível de usuário. A Figura 3.5 representa a tela de acesso do administrador e a Figura 3.6 representa a tela de acesso do aluno.

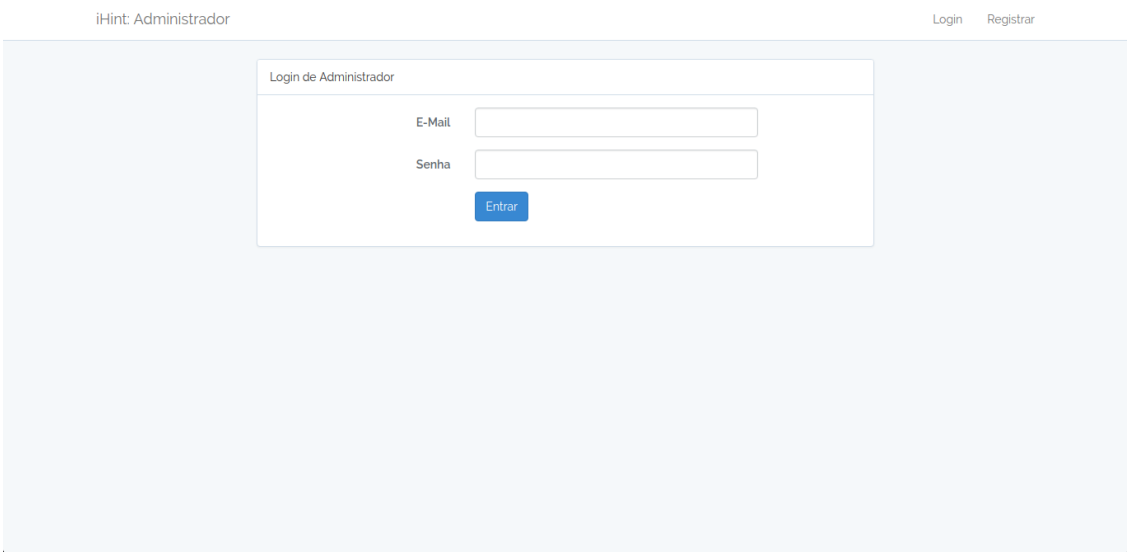

Figura 3.5. Acesso do Administrador.

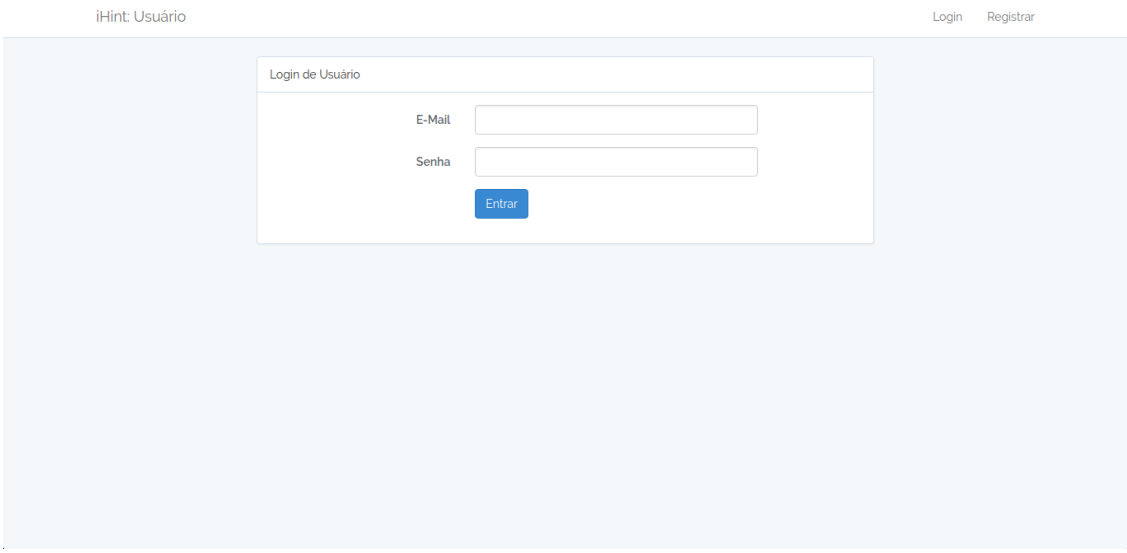

Figura 3.6. Acesso do Aluno.

Cadastro: módulo responsável pelo cadastro do usuário, no qual são coletados os dados do usuário como aluno ou como administrador do sistema. No caso do administrador, o sistema exige que cadastre um nome, e-mail e senha para finalizar o cadastro. Para o papel de aluno, o usuário deve informar o nome, e-mail, universidade, curso, semestre, experiência com programação antes da graduação, se trabalha ou trabalhou com programação e a senha. A Figura 3.7 representa a tela de cadastro do administrador e a Figura 3.8 representa a tela de cadastro do aluno.

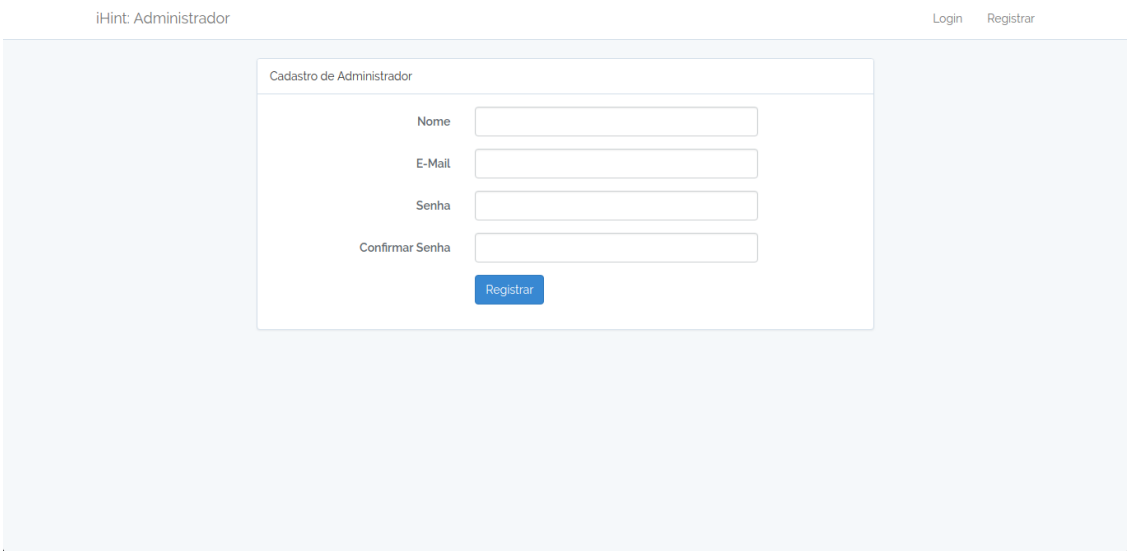

Figura 3.7. Cadastro do Administrador.

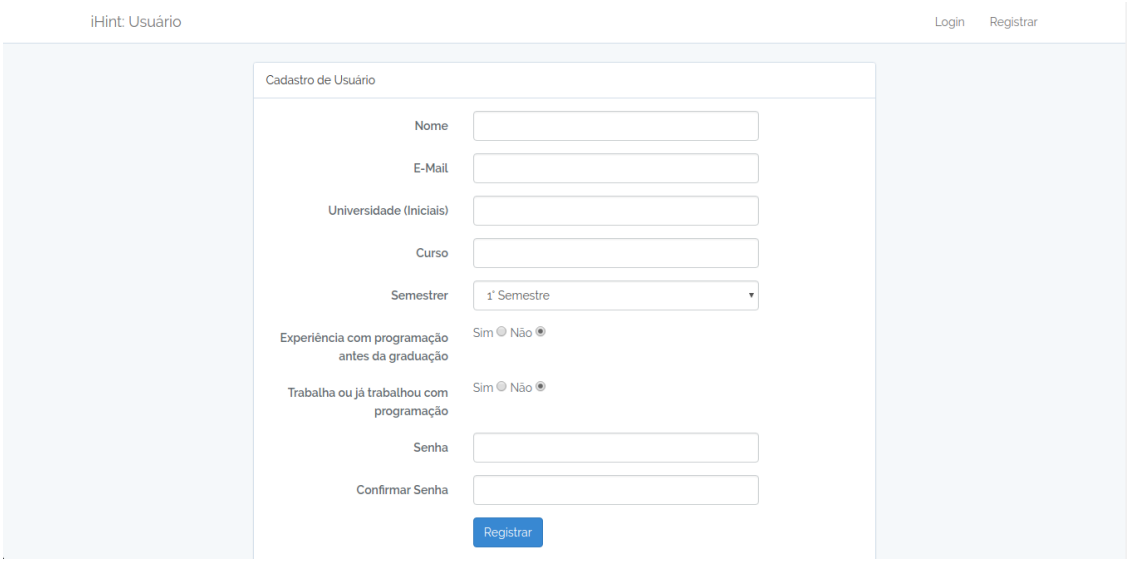

Figura 3.8. Cadastro do Aluno.

Após autenticação no sistema, o usuário habilitado pode navegar através do menu principal, que contém o fluxo operacional do sistema. O fluxo irá depender do tipo do usuário: administrador, representador pela Figura 3.9, ou aluno, representado pela Figura 3.10.

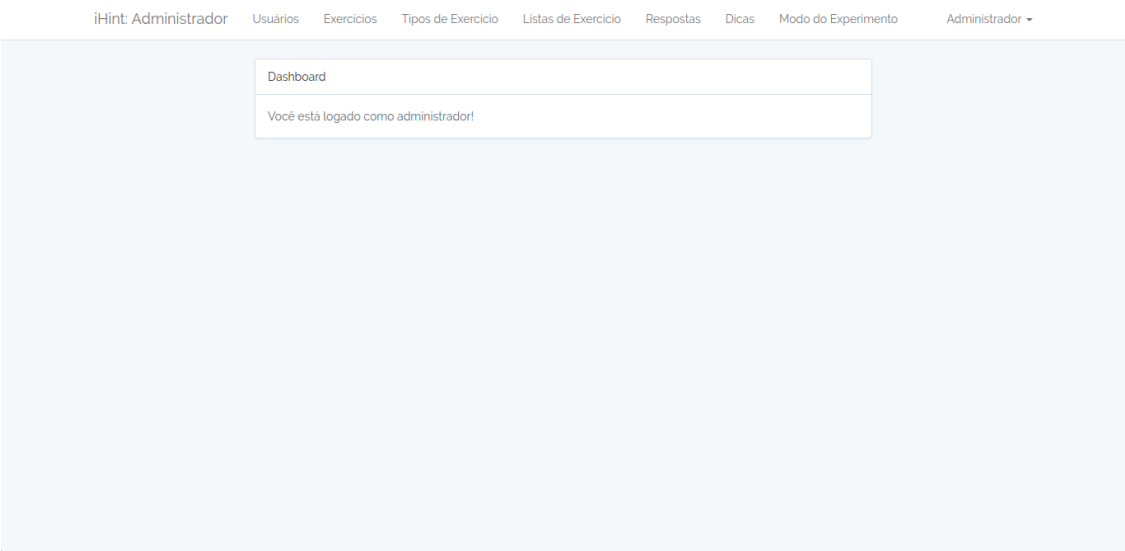

Figura 3.9. Dashboard do Administrador.

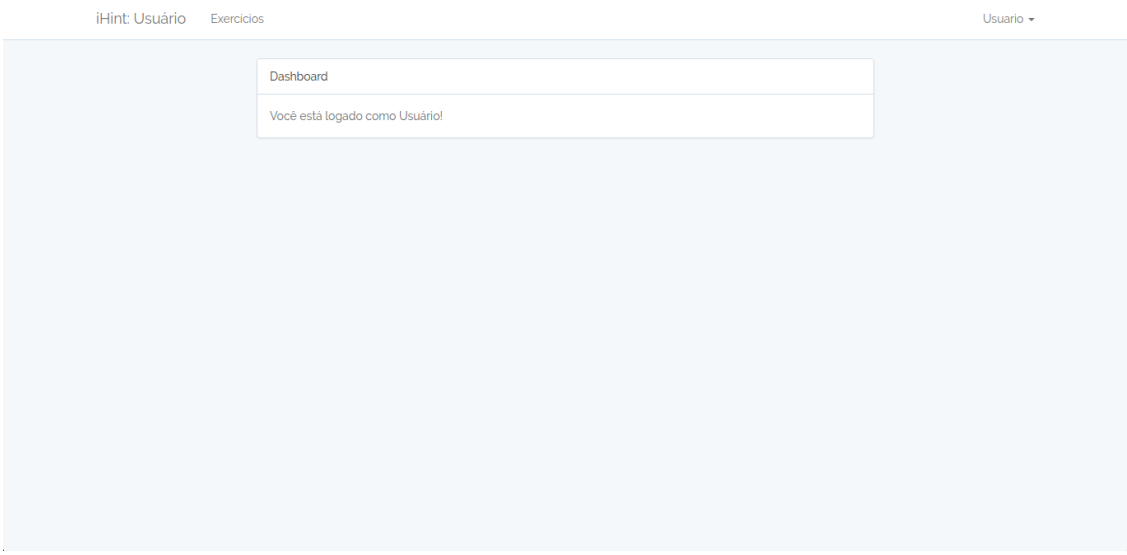

Figura 3.10. Dashboard do Aluno.

3.2.1. Atividades do Administrador

O administrador gerencia os usuários do sistema, podendo consultar seus dados e realiza o gerenciamento de exercícios, tipos de exercícios a serem disponibilizados e listas de exercícios a serem oferecidas aos alunos. Além disso, ele tem acesso às respostas e dicas criadas pelos alunos no sistema.

Gerenciar Usuários: A Figura 3.11 apresenta a lista de usuários cadastrados no sistema. O administrador tem acesso aos dados de cada usuário clicando no campo “Id”.

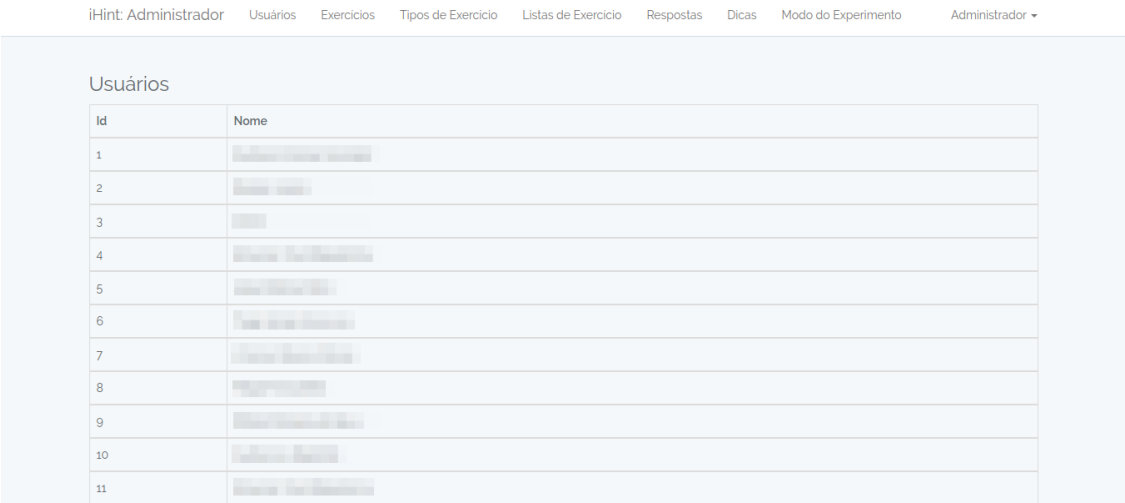

Figura 3.11. Lista de Usuários.

Gerenciar Exercícios: No módulo de exercícios, o administrador acessa aos exercícios cadastrados no sistema, apresentada pela Figura 3.12, que mostra o enunciado, tipo do exercício e sua resposta.

Figura 3.12. Exercícios Cadastrados.

Caso queira cadastrar um novo exercício, é preciso clicar no botão “Novo Exercício”, que redireciona o administrador para a tela de cadastro de exercício. A Figura 3.13 exibe o que é necessário para criar um novo exercício: enunciado, tipo do exercício, respostas geradas a partir de uma entrada que a submissão deve processar. No caso, é permitida a inserção de várias entradas e respostas, permitindo a avaliação automática e meticulosa da correção da submissão do aluno.

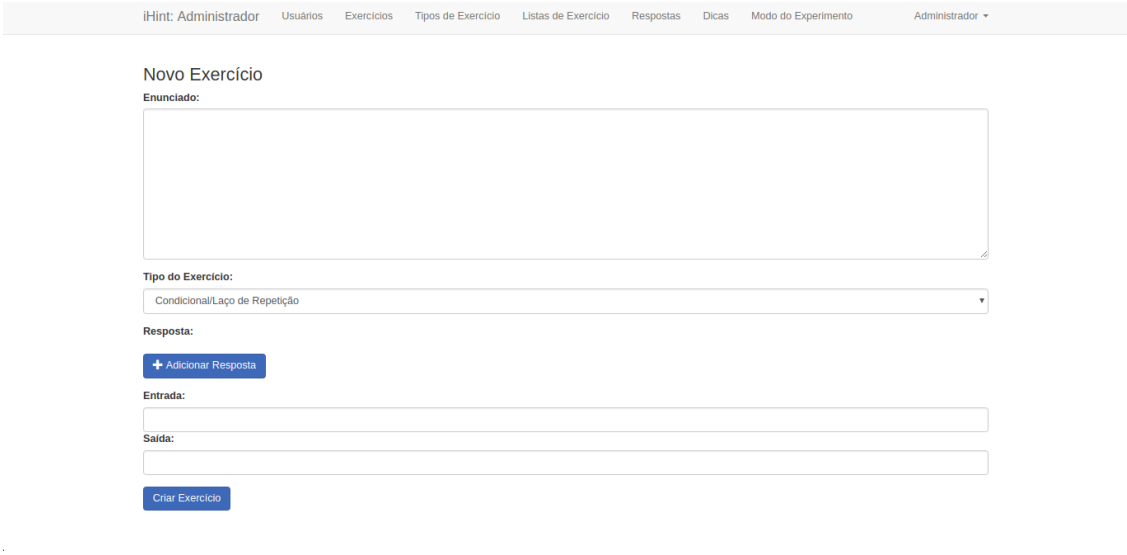

Figura 3.13. Cadastro de Exercício.

Gerenciar Lista de Exercícios: Cada exercício cadastrado pode ser agregado a uma lista de exercícios que o administrador tenha cadastrado anteriormente, assim ele consegue criar listas de exercícios personalizadas. A Figura 3.14 mostra a tela que apresenta todas as listas criadas pelo administrador.

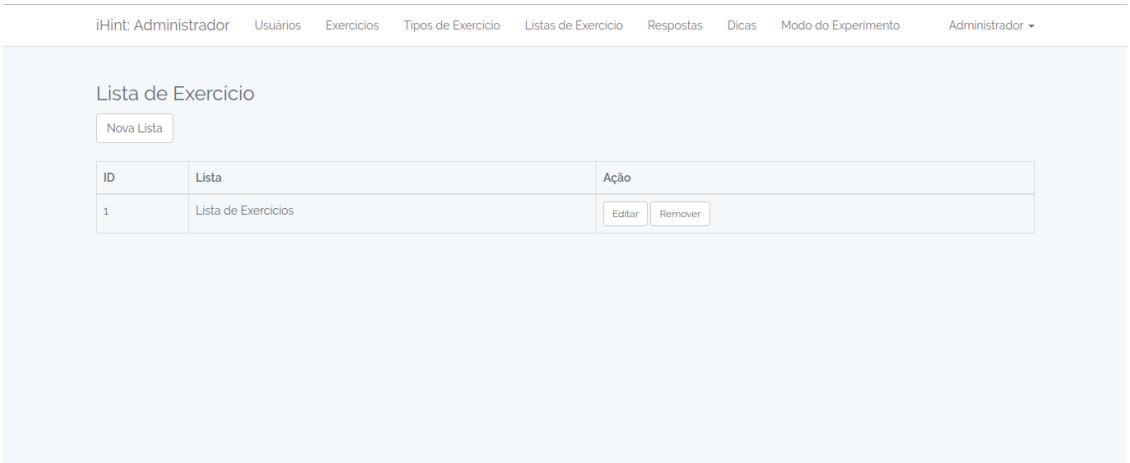

Figura 3.14. Listas de Exercícios Cadastradas.

Na Figura 3.15 é apresentada a tela em que o administrador realiza o cadastro de uma lista de exercícios, informando o nome da lista e os exercícios que deseja adicionar. Os exercícios são adicionados a partir da consulta no banco de dados de exercícios: o sistema retorna uma coleção de exercícios cadastrados no sistema, da qual o administrador escolhe os exercícios que deseja adicionar na lista.

Exercícios

ID	Exercício	Ação
1	Joaozinho quer calcular e mostrar a quantidade de litros de combustível gastos em uma viagem, ao utilizar um automóvel que faz 12 KM/L. Para isso, ele gostaria que você o auxiliasse através de um simples programa. Para efetuar o cálculo, deve-se fornecer o tempo gasto na viagem (em horas) e a velocidade média durante a mesma (em km/h). Assim, pode-se obter distância percorrida e, em seguida, calcular quantos litros seriam necessários. Mostre o valor com 3 casas decimais após o ponto. Entrada: A entrada contém dois inteiros. O primeiro é o tempo gasto na viagem (em horas) e o segundo é a velocidade média durante a mesma (em km/h). Saída: Imprima a quantidade de litros necessária para realizar a viagem, com três dígitos após o ponto decimal Exemplo de Entrada: 10 85 Exemplo de Saída: 70.833	Adicionar na Lista
2	Leia quatro valores inteiros A, B, C e D. A seguir, calcule e mostre a diferença do produto de A e B pelo produto de C e D segundo a fórmula: DIFERENCA = (A * B - C * D). Entrada: A entrada contém 4 valores inteiros. Saída: Imprima a mensagem DIFERENCA com todas as letras maiúsculas, conforme exemplo abaixo, com um espaço em branco antes e depois da igualdade. Exemplos de Entrada: 5 6 7 8 Exemplos de Saída: DIFERENCA = -26	Adicionar na Lista
3	Leia 4 valores inteiros A, B, C e D. A seguir, se B for maior do que C e se D for maior do que A, e a soma de C com D for maior que a soma de A e B e se C e D, ambos, forem positivos e se a variável A for par escrever a mensagem "Valores aceitos", senão escrever "Valores nao aceitos". Entrada: Quatro números inteiros A, B, C e D. Saída: Mostre a respectiva mensagem após a validação dos valores. Exemplo de Entrada: 5 6 7 8 Exemplo de Saída: Valores nao aceitos	Adicionar na Lista
4	Leia um valor inteiro correspondente à idade de uma pessoa em dias e informe-a em anos, meses e dias Obs.: apenas para facilitar o cálculo, considere todo ano com 365 dias e todo mês com 30 dias. Nos casos de teste nunca haverá uma situação que permite 12 meses e alguns dias, como 360, 363 ou 364. Este é apenas um exercício com objetivo de testar raciocínio matemático simples. Entrada: A entrada contém um valor inteiro. Saída: Imprima a saída conforme exemplo fornecido. Exemplo de Entrada: 400 Exemplo de Saída: 1 ano(s) 1 mes(es) 5 dia(s)	Adicionar na Lista
5	Leia um valor inteiro, que é o tempo de duração em segundos de um determinado evento em uma fábrica, e informe-o expresso no formato horas:minutos:segundos. Entrada: A entrada contém um valor inteiro N. Saída: Imprima o tempo lido na entrada (segundos), convertido para horas:minutos:segundos, conforme exemplo fornecido. Exemplo de Entrada: 556 Exemplo de Saída: 0:9:16	Adicionar na Lista
6	Leia 3 valores de ponto flutuante e efetue o cálculo das raízes da equação de Bhaskara. Se não for possível calcular as raízes, mostre a mensagem correspondente "Impossível calcular", caso haja uma divisão por 0 ou raiz de número negativo. Entrada: Leia três valores de ponto flutuante (double) A, B e C. Saída: Se não houver possibilidade de calcular as raízes, apresente a mensagem "Impossível calcular". Caso contrário, imprima o resultado das raízes com 5 dígitos após o ponto, com uma mensagem correspondente conforme exemplo abaixo. Imprima sempre o final de linha após cada mensagem. Exemplos de Entrada: 10.0 20.1 5.1 Exemplos de Saída: R1 = -0.29788 R2 = -1.71212	Adicionar na Lista
7	Faça um programa que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o total a receber no final do mês, com duas casas decimais. Entrada: A entrada contém um texto (primeiro nome do vendedor) e 2 valores de dupla precisão (double) com duas casas decimais, representando o salário fixo do vendedor e montante total das vendas efetuadas por este vendedor, respectivamente. Saída: Imprima o total que o funcionário deverá receber, conforme exemplo fornecido. Exemplos de Entrada: JOAO 500.00 1230.30 Exemplos de Saída: TOTAL = R$ 684.54	Adicionar na Lista
8	Leia um valor de ponto flutuante com duas casas decimais. Este valor representa um valor monetário. A seguir, calcule o menor número de notas e moedas possíveis no qual o valor pode ser decomposto. As notas consideradas são de 100, 50, 20, 10, 5, 2. As moedas possíveis são de 1, 0.50, 0.25, 0.10, 0.05 e 0.01. A seguir mostre a relação de notas necessárias. Entrada: A entrada contém um valor de ponto flutuante N (0 ≤ N ≤ 1000000.00). Saída: Imprima a quantidade mínima de notas e moedas necessárias para trocar o valor inicial, conforme exemplo fornecido. Exemplo de Entrada: 576.73 Exemplo de Saída: NOTAS: 5 nota(s) de R$ 100.00 1 nota(s) de R$ 50.00 1 nota(s) de R$ 20.00 0 nota(s) de R$ 10.00 1 nota(s) de R$ 5.00 0 nota(s) de R$ 2.00 MOEDAS: 1 moeda(s) de R$ 1.00 1 moeda(s) de R$ 0.50 0 moeda(s) de R$ 0.25 2 moeda(s) de R$ 0.10 0 moeda(s) de R$ 0.05 3 moeda(s) de R$ 0.01	Adicionar na Lista
9	Joaozinho quer calcular e mostrar a quantidade de litros de combustível gastos em uma viagem, ao utilizar um automóvel que faz 12 KM/L. Para isso, ele gostaria que você o auxiliasse através de um simples programa. Para efetuar o cálculo, deve-se fornecer o tempo gasto na viagem (em horas) e a velocidade média durante a mesma (em km/h). Assim, pode-se obter distância percorrida e, em seguida, calcular quantos litros seriam necessários. Mostre o valor com 3 casas decimais após o ponto. Entrada: A entrada contém dois inteiros. O primeiro é o tempo gasto na viagem (em horas) e o segundo é a velocidade média durante a mesma (em km/h). Saída: Imprima a quantidade de litros necessária para realizar a viagem, com três dígitos após o ponto decimal Exemplo de Entrada: 10 85 Exemplo de Saída: 70.833	Adicionar na Lista

Nome da Lista:

Figura 3.15. Cadastrar Lista.

Consultar Respostas: O administrador é capaz de visualizar todas as respostas submetidas no sistema de dicas. A Figura 3.16 indica os exercícios submetidos pelos alunos, apresentando o caminho em que a resolução do aluno foi armazenada no sistema.

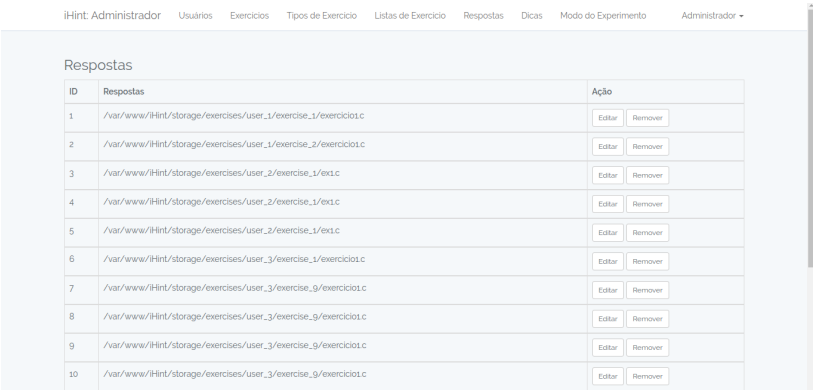

Figura 3.16. Lista de Respostas.

Consultar Dicas: O administrador pode realizar a mudança do tipo de experimento através do menu "Modo do Experimento" visível na Figura 3.9, os modos disponíveis são:

Modo Coleta: O sistema possui dois modos de operação, permitindo a realização de estudos acadêmicos ao controlar o uso e avaliação de dicas. No Modo Coleta, os alunos preenchem a base de dicas para que, posteriormente, no Modo Experimento, os alunos que estiverem utilizando o sistema possam ter acesso a algumas dicas para realizar os exercícios. Neste modo, os alunos não recebem nem avaliam dicas.

Modo Experimento: O sistema funciona na sua forma original, em que os alunos resolvem os exercícios, utilizando dicas para auxiliar na realização dos exercícios, e criam novas dicas para o sistema.

3.2.2. Atividades do Aluno

O aluno apenas tem acesso ao módulo de exercícios, em que realiza exercícios, submetendo soluções e usando dicas, e cria dicas para o sistema.

Utilizar Lista de Exercícios: módulo que apresenta a lista de exercícios cadastrada pelo administrador, representada pela Figura 3.17. O aluno pode escolher qualquer exercício da lista para realizar.

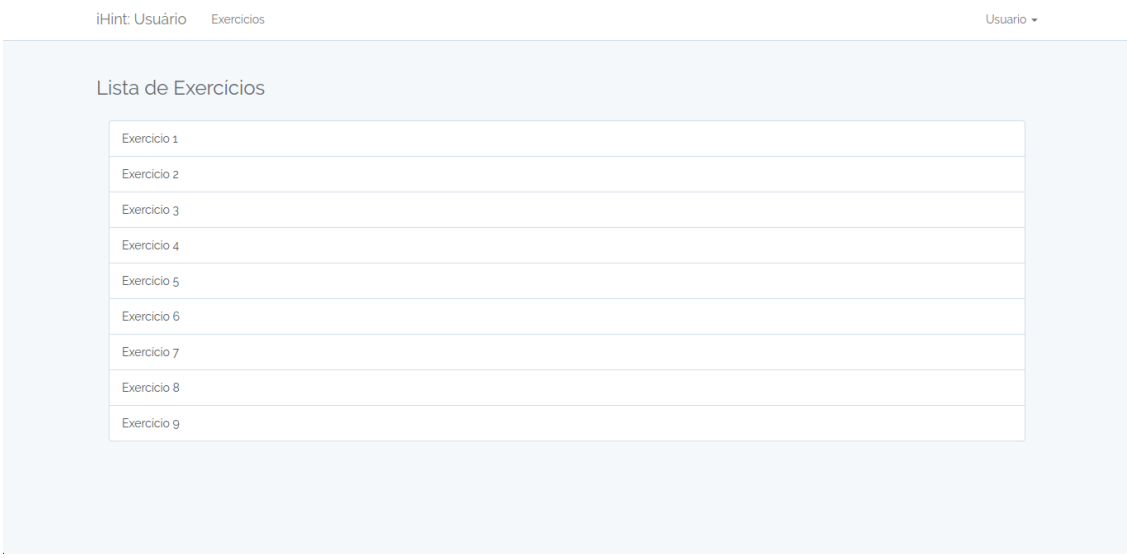

Figura 3.17. Lista de Exercícios.

Realizar Exercício: Após a escolha do exercício, o sistema apresenta a tela em que o aluno consegue realizar o exercício, apresentada pela Figura 3.18. Nesta tela o usuário tem acesso ao enunciado e ao formulário para envio de uma solução, o que deve ser efetuado através do envio (*upload*) de um arquivo na linguagem C para resolução do exercício.

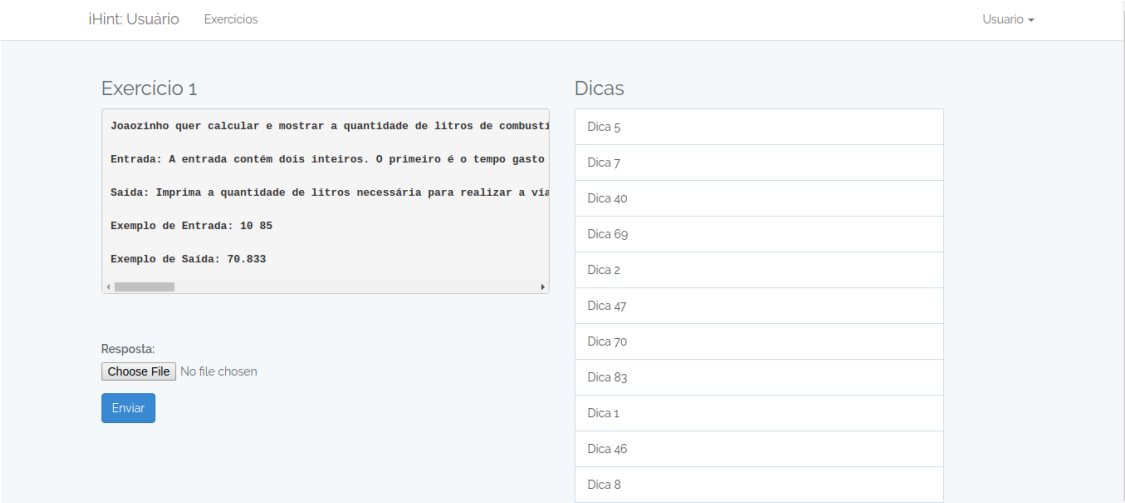

Figura 3.18. Realizar Exercício.

Conjuntamente à Figura 3.18, o aluno tem acesso às dicas produzidas por outros alunos para o exercício escolhido, podendo utilizá-las em todas as submissões que realizar até que resolva o exercício.

Em seguida, o usuário pode trabalhar em sua solução e submeter ao sistema para correção. Se estiver correta, o sistema redireciona o usuário para tela de cadastro da dica, mostrado na Figura 3.19. Caso a resolução esteja errada, o sistema retorna para a tela do exercício para que o

usuário possa realizar outras submissões até que consiga obter sucesso na resolução ou requisitar uma dica para o sistema auxiliá-lo na construção da resposta.

As correções dos exercícios são realizadas automaticamente, utilizando-se as respostas cadastradas para cada exercício, conforme ilustrado na Figura 3.13. Para cada exercício, podem estar cadastradas mais de uma tupla de entrada e saída para testar as soluções enviadas pelos alunos. O sistema utiliza o método **exec** do PHP, que possibilita a execução de programas externos. Assim, para realizar a validação do exercício, é executado o comando *GNU Compiler Collection (GCC)* para compilar o exercício submetido, cujo resultado é executado com as entradas cadastradas, comparando-se o retorno com a resposta cadastrada no exercício.

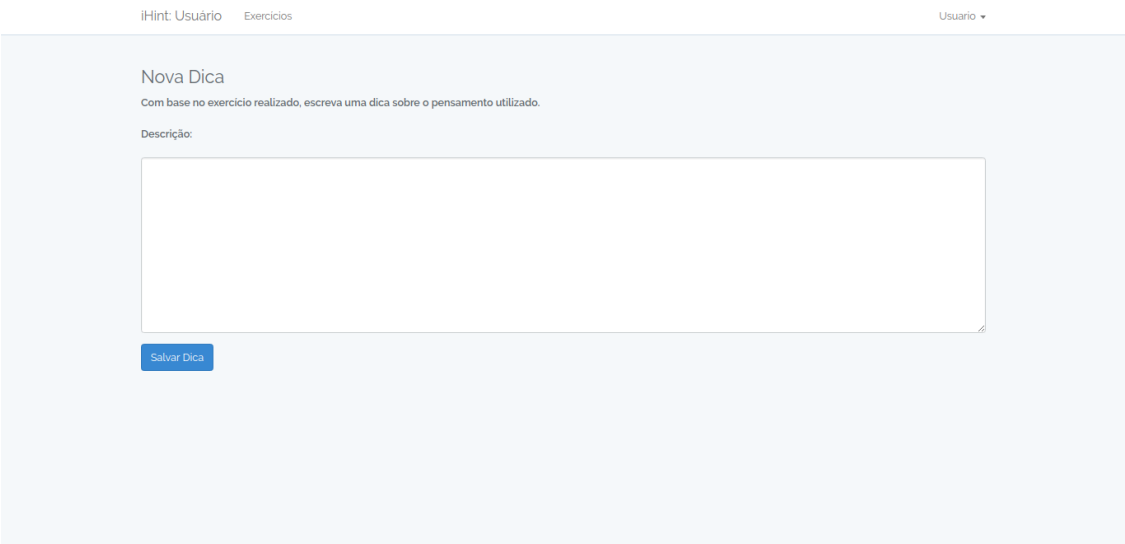

Figura 3.19. Cadastrar Dica.

Cada submissão, tanto correta ou incorreta, é armazenada no banco de dados, sendo registrados os dados: identificador do usuário, identificador do exercício, validação da resposta (Verdadeiro ou Falso), código submetido pelo usuário, número de linhas da resposta, complexidade ciclomática, data e hora da submissão.

Utilizar Dicas: Quando o aluno requisitar uma dica, clicando em uma dica da lista apresentada (Figura 3.18), o sistema exibe a conteúdo da dica. Após sua leitura, o aluno realiza a classificação da dica, conforme exemplo exibido na Figura 3.20.

Figura 3.20. Utilizar a Dica.

3.2.3. Exemplo de Código

Para exemplificar o código do sistema de dica implementado com o *framework* Laravel, foi escolhida a funcionalidade apresentada na Figuras 3.14 e 3.15, em que o administrador realiza o gerenciamento de lista de exercícios. Para agilizar o desenvolvimento, o Laravel possui uma ferramenta de linha de comando chamada *Artisan*, que trata da geração das classes disponibilizadas pelo framework. A seguir são mostradas as etapas realizadas para o desenvolvimento dessa funcionalidade.

A primeira etapa consiste em criar o modelo e o repositório da nossa aplicação. Primeiramente cria-se a tabela do banco de dados, executando-se o comando **php artisan make:migration create_exerciselists_table**. Neste caso, a classe de migração *CreateExerciselistsTable* é criada, possibilitando a configuração do *schema* da tabela do banco de dados utilizando a linguagem PHP, conforme mostrado no Código-fonte 3.1. Nesse exemplo, são criados os atributos *id*, *topic* e *exercises*. Observa-se que todo código gerado pelo *Artisan* está no padrão de escrita PSR-2 facilitando a manutenção e entendimento do código. Posteriormente, executa-se o comando **php artisan migrate** para sincronizar a classe e o esquema do banco de dados.

```
class CreateExerciselistsTable extends Migration
{

  public function up()
  {
    Schema::create('exercise_lists', function (Blueprint $table) {
      $table->increments('id');
      $table->string('topic');
      $table->string('exercises');
      $table->timestamps();
    });
  }

  public function down()
```

```
    {
      Schema::dropIfExists('exercise_lists');
    }
  }
```

Código-fonte 3.1. Migração Gerada

Na segunda etapa, é criado o modelo para a tabela *exercise_lists*, executando o comando **php artisan make:model ExerciseList**. O mapeamento entre o modelo e a migração é realizado pelo nome do modelo informado. Por padrão, o Laravel assume que a migração deve ser criada no plural e o modelo no singular. O Código-fonte 3.2 apresenta o modelo criado, em que é necessário criar uma lista com os nomes dos campos descritos na migração, permitindo que o ORM atribua os valores que serão inseridos na tabela.

```
class ExerciseList extends Model implements Transformable
{
  use TransformableTrait;

  protected $fillable = [
    'topic',
    'exercises'
  ];

}
```

Código-fonte 3.2. Modelo Gerado

Com o modelo criado, para utilizar o *Repository Design Pattern* é preciso criar um repositório no Laravel utilizando o comando **php artisan make:repository ExerciseListRepository**, que gera o Código-fonte 3.3, no qual realiza o mapeamento do modelo através da função *model* da classe *ExerciseListRepositoryEloquent*.

```
class ExerciseListRepositoryEloquent extends BaseRepository implements ExerciseListRepository
{
  public function model()
  {
    return ExerciseList::class;
  }
  public function boot()
  {
    $this->pushCriteria(app(RequestCriteria::class));
  }
}
```

Código-fonte 3.3. Repositório Gerado

A terceira etapa consiste na criação da classe de controle que realiza a comunicação da *view* com o *model*, valida e envia os dados para os modelos correspondentes. Foi realizada a criação do modelo para gerenciar o fluxo de dados da funcionalidade, executando o comando **php artisan make:controller ExerciseListController**.

```
class ExerciseListController extends Controller
{

  private $repository;

  public function __construct(ExerciseListRepository $repository)
  {
    $this->repository = $repository;
  }

  public function index()
  {
    $exerciselists = $this->repository->paginate();

    return view('admin.exerciselist.index', compact('exerciselists'));
  }

  public function store(AdminExerciseListRequest $request)
  {
    $data = $request->all();

    $this->repository->create($data);

    return redirect()->route('admin.exerciselist.index');
  }

.
.
.
```

Código-fonte 3.4. Controle Gerado

O Código-fonte 3.4 mostra apenas a criação do construtor do repositório, da função *index* que consulta no banco as listas cadastradas, utilizando o método *paginate* para realizar a paginação dos dados, e da função *store*, que valida os dados através da função *AdminExerciseListRequest*, recuperando todos os dados da requisição utilizando o método *all* e, por fim, realizando a inserção no banco de dados com o método *create* disponibilizada pelo construtor do repositório.

Na quarta e última etapa, é necessária a criação da *view*, utilizando-se a *Engine* de *template* Blade. Nesse caso, o Laravel não disponibiliza um comando para a geração de *templates*, pois a utilização do Blade não é obrigatória. O Código-fonte 3.5 mostra que o Blade permite a inserção de HTML utilizando o comando *@extends*. Desta forma é possível reutilizar códigos em todas as telas do sistema. Também é possível adicionar formulários e botões utilizando o *Form* disponibilizado pelo Laravel, que permite criar todos os tipos de *inputs* que o HTML suporta.

```
@extends('admin.layout.auth')

@section('content')
  <div class="container">
    <h3>Exercicios</h3>

    @include('errors._check')
```

```
        {!! Form::open(['route'=>'admin.exerciselist.store', 'class'=>'form']) !!}

        @include('admin.exerciselist._form')

        <div class="form-group">
          {!! Form::submit('Criar Lista', ['class'=>'btn btn-primary']) !!}
        </div>

        {!! Form::close() !!}
      </div>
    @endsection
```

Código-fonte 3.5. *View* Gerada

Finalmente, no *Form::open* é preciso passar a rota que mapeia a função *store* que está no controller *ExerciseListController*. Com isso, quando o usuário clicar no botão para realizar o cadastro, o Laravel faz uma requisição *POST* com os dados que estão no formulário para a rota especificada. Desse modo, o controlador reconhece a requisição, passa pela função que realiza a validação e armazena no banco.

3.3. Considerações Finais

Neste capítulo foi apresentado o desenvolvimento do sistema de dicas. Foi explicado sobre o *Framework* utilizado, suas características e arquitetura. Também foram apresentadas as funcionalidades desenvolvidas e como seria o fluxo de atividades realizadas por cada usuário. A próxima seção mostrará os estudos realizados e os resultados obtidos pela abordagem proposta neste capítulo.

Capítulo 4

Estudo, Resultados e Discussão

Este capítulo descreve o estudo realizado para avaliar o sistema de dicas descrito neste trabalho. A Seção 4.1 apresenta como foram executadas as etapas do estudo. Na Seção 4.2 são mostrados os resultados obtidos e a discussão.

4.1. Estudo

Após a implementação do sistema de dicas apresentada no Capítulo 3, foi realizado um estudo para avaliar o sistema. O primeiro passo foi a definição de uma lista de exercícios. Foi preciso preencher a base de dados de exercícios a partir de exercícios selecionados para diferir daqueles realizados na disciplina de algoritmos oferecida pelo curso de BCC da UTFPR-CM. A seleção teve a finalidade de prevenir que os participantes do estudo já tenham realizado os exercícios. Chegou-se ao conjunto de exercícios mostrado na Apêndice B.

Após o preenchimento da base de exercícios, prosseguiu-se para a realização do estudo, organizado em três etapas, conforme apresentado na Tabela 4.1. As primeiras etapas tiveram como objetivo a criação de dicas para o sistema, permitindo a avaliação do sistema implementado em um cenário em que existe um conjunto útil, porém limitado, de dicas. Estas etapas foram executadas no Modo Coleta da ferramenta. Na terceira etapa, os alunos podiam resolver exercícios, usar, criar e avaliar dicas, utilizando integralmente as funcionalidades do sistema.

Tabela 4.1. Voluntários do Estudo.

Estudo	Quantidade de Voluntários	Semetre
1ª Etapa do Estudo	6	8
2ª Etapa do Estudo	3	-
3ª Etapa do Estudo	7	1
Total:	16	-

Todas as etapas tiveram duração média de 3 horas. No início de cada etapa, foi aplicado o termo de consentimento, apresentado no Apêndice A, para que os participantes ficassem cientes sobre o uso dos dados que seriam gerados durante a utilização do sistema e sobre a garantia de anonimato dos dados.

A primeira etapa ocorreu na UTFPR-CM, em meados do primeiro semestre de 2017, com a participação de 6 estudantes do oitavo período do curso de BCC. A segunda etapa foi realizada com 3 profissionais da indústria na área de Computação, no meio do segundo semestre de 2017. Por fim, a terceira etapa aconteceu em dois dias distintos no mês de outubro de 2017, com alunos do primeiro período do curso de BCC da UTFPR-CM.

A primeira e segunda etapa do estudo utilizaram o "Modo Coleta" do sistema apresentado na Seção 3.2.1, pois esses estudos tiveram o objetivo de isolar a criação das dicas realizadas pelos participantes. Assim foi possível dividir as dicas criadas por acadêmicos e profissionais da indústria da área da computação, para que, após a última etapa, pudesse ser realizada a comparação do desempenho das dicas desses dois tipos de participantes.

No final da terceira etapa, utilizou-se o "Modo Experimento" do sistema descrito na Seção 3.2.1, onde os participantes puderam realizar os exercícios, consultar e criar dicas. Os alunos puderam consultar dicas criadas na primeira e segunda etapa do estudo, mas também tiveram acesso às dicas criadas por outros participantes da terceira etapa que estavam realizando a lista de exercício juntamente. No término da terceira etapa, foi aplicado um questionário para os participantes avaliarem a utilização do sistema e a importância que as dicas tiveram durante a execução dos exercícios.

Na Figura 4.1 é apresentado o fluxo que os participantes seguiram no sistema para realizar a solução de exercícios e a criação das dicas. O fluxo é iniciado quando o usuário recebe uma lista de exercícios para ser realizada. O sistema apresenta um exercício da lista para ser resolvido e o usuário pode começar a realizar a solução. No caso da terceira etapa, o voluntário pode consultar algumas dicas. Deste modo, o sistema apresenta as dicas disponíveis para o exercício que está sendo realizado, as quais o usuário pode consultar para realizar sua solução. Cabe destacar que, para as etapas 1 e 2 deste estudo, a funcionalidade de consulta às dicas está desabilitada (Modo Coleta).

Após a solução ser concluída, o usuário submete a solução para validação, o sistema executa a solução do exercício e verifica se está correta ou não. Caso a solução não esteja correta, o sistema redireciona para a etapa de realização de solução para que o participante possa realizar uma nova tentativa. Caso a solução esteja correta, o sistema redireciona o usuário para a tela em que ele pode realizar a criação e cadastro da dica no banco de dicas.

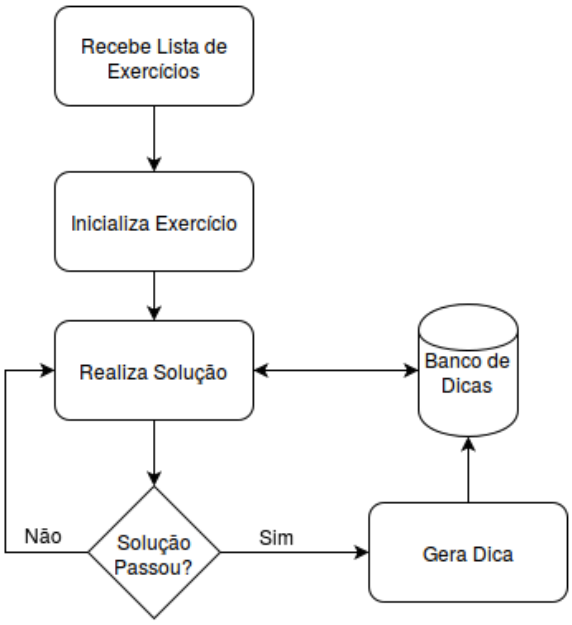

Figura 4.1. Fluxo para Criação das Dicas.
Fonte: Adaptado de Glassman et al. (2016).

4.2. Resultados

Após a execução do estudo, realizou-se a análise a partir dos dados capturados nas submissões e dicas utilizadas pelos participantes. No total, foram realizadas 243 submissões de exercícios no sistema de dicas, sendo que 191 ocasionaram em erros e 52 em acertos. Conforme mencionado anteriormente, para cada solução correta efetuada, os alunos podiam criar dicas referentes ao exercício submetido. A Tabela 4.2 mostra a quantidade de dicas criadas em cada etapa do estudo realizado.

Tabela 4.2. Dicas Criadas por Etapa.

Etapas	Quantidade de Dicas Criadas
1° Etapa	40
2° Etapa	21
3° Etapa	18
Total	79

A Tabela 4.3 apresenta a quantidade de dicas criadas por exercício.

Tabela 4.3. Dicas Criadas por Exercício.

Exercício	Quantidade de Dicas Criadas
1° Exercício	13
2° Exercício	12
3° Exercício	12
4° Exercício	12
5° Exercício	11
6° Exercício	9
7° Exercício	10
Total	79

A Figura 4.2 apresenta as submissões, no caso separadas pelas etapas do estudo executado. A primeira etapa, em que ocorreu a criação de dicas, obteve um total de 136 envios de exercícios, contendo 47 tentativas certas e 89 erradas. Na segunda etapa de criação de dicas, os usuários realizaram 43 submissões, com 24 certas e 19 erradas. Nas primeira e segunda etapas, os usuários não utilizaram a funcionalidade de consultar dicas, pois o objetivo dessas etapas eram apenas criar as dicas que seriam utilizadas futuramente na terceira etapa. Na última etapa, os usuários realizaram os exercícios com o apoio das dicas, tem um total de 64 submissões, contendo 28 certas e 36 erradas.

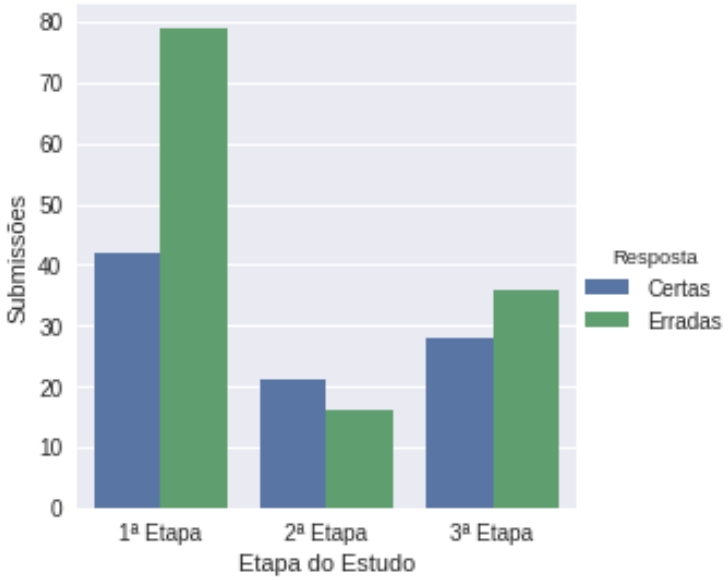

Figura 4.2. Visão Geral das Submissões Realizadas.

A Figura 4.2 apresenta indícios que os usuários da terceira etapa tiveram maior facilidade em resolver os exercícios pelo fato da utilização das dicas, comparando os resultados da primeira e terceira etapa. A segunda etapa obteve o melhor resultado mesmo sem utilizar as dicas, mas

isso pode estar relacionado ao fato dos participantes dessa etapa serem profissionais da área de computação que apresentam maior experiência com a resolução de problemas.

4.2.1. Submissões Realizadas

Efetuou-se a separação dos dados para apresentar o total de submissões efetuadas pelos participantes em cada exercício. Inicialmente, nas etapas do estudo apresentado na Figura 4.3, a média das submissões nas etapas 1 e 2, nas quais os usuários não consultaram dicas para resolver os exercícios, foi de 15,29 e 5,28 respectivamente.

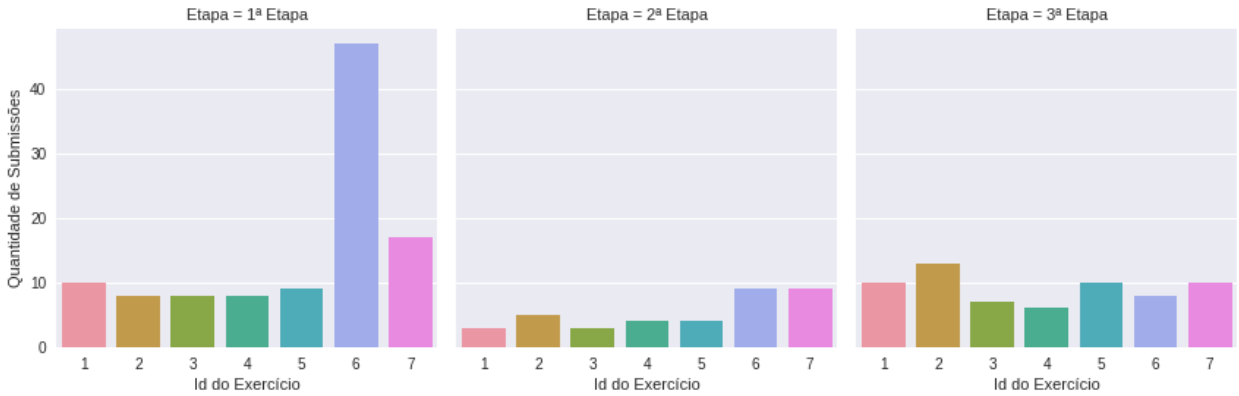

Figura 4.3. Submissões por Etapas do Estudo.

Posteriormente, na terceira etapa, a média de submissões foi de 9,14 por exercício. Em relação aos resultados da primeira etapa, com alunos experientes, é possível encontrar indicativos que a utilização das dicas para resolver exercícios no sistema ajudou os participantes da terceira etapa. Cabe observar ainda que, para a terceira etapa, não foi possível realizar a aplicar a terceira etapa do estudo com uma turma completa do primeiro semestre da disciplina de algoritmos e, portanto, analisar os resultados quanto à participação de alunos de baixo, médio e alto desempenho.

Na Figura 4.4 é mostrada a comparação entre a quantidade de submissões certas e erradas realizadas nas três etapas do estudo. Embora os primeiros exercícios sejam mais simples e consequentemente sujeitos a uma quantidade menor de submissões incorretas, é possível observar que eles apresentam uma quantidade elevada de tentativas erradas entre os alunos. Isso se deve, em parte, à necessidade de adaptação dos usuários para começar a utilizar o sistema, pois os usuários tiveram que adequar a forma de compilação dos exercícios que o sistema utilizava e o padrão de construção da resposta esperado pelo avaliador automático de correção do sistema.

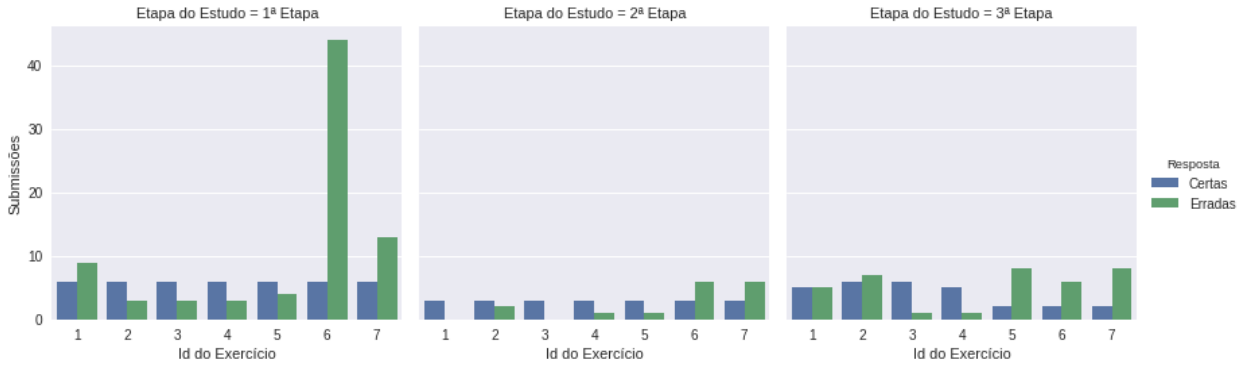

Figura 4.4. Submissões Certas e Erradas por Etapa.

Observa-se também que o exercício 6 apresentou um número muito elevado de submissões na primeira etapa. Investigando esse comportamento, foi possível apontar que a maior dificuldade dos participantes estava relacionada com uma das regras que o exercício necessitava: a maioria do pensamento construído pelos participantes para realizar o exercício estava correta, mas uma regra não estava bem desenvolvida no código, exigindo vários ajustes e testes para que o exercício ficasse correto. No caso do exercício 7, a linguagem C realizava um arredondamento ao utilizar o conversor *atof*, que realiza a conversão de uma *string* para um valor flutuante, o que acabava atrapalhando nas operações.

Na segunda etapa, os participantes não tiveram problemas na realização dos exercícios. Mas nos exercícios 6 e 7, passaram pelos mesmos problemas dos participantes da primeira etapa.

Por fim, na terceira etapa os usuários puderam acessar as dicas criadas pelos participantes da primeira e segunda etapa que passaram pelos problemas citados. No entanto, apenas os usuários 13 e 14 realizaram os exercícios 6 e 7, não sendo possível verificar se as dicas ajudaram no desenvolvimento desses exercícios.

4.2.2. Submissões Erradas

Aprofundando na análise das submissões, na Figura 4.5 foi feito um agrupamento dos dados da primeira, segunda e terceira etapa de modo a exibir a relação entre a quantidade de submissões erradas por usuário em cada exercício, com o objetivo de investigar em quais exercícios os participantes apresentaram maior dificuldade.

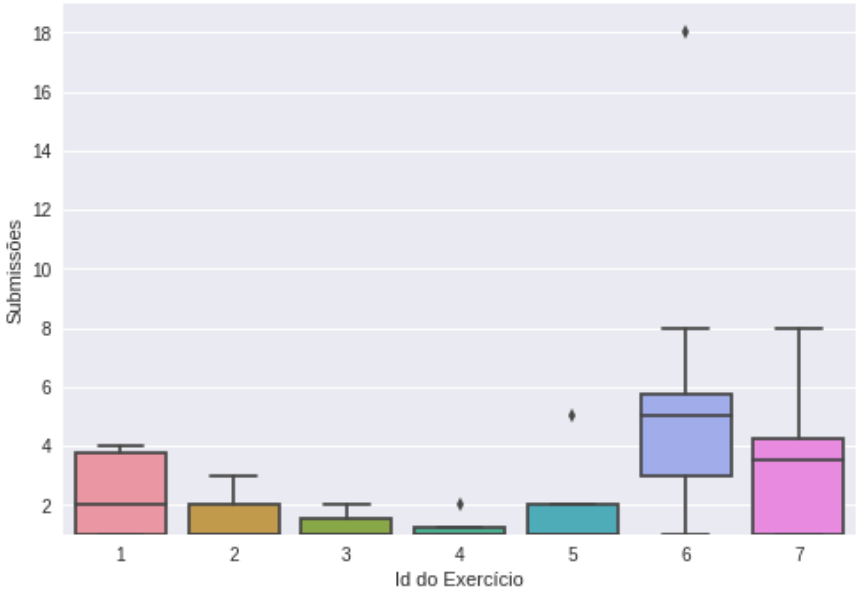

Figura 4.5. Submissões Erradas nas Etapas.

O primeiro exercício realizado nos estudos foi o que apresentou a maior concentração de participantes, sendo realizado por todos os usuários. Verificou-se que, nesse exercício, o número de tentativas erradas se concentrou entre 1 e 4 por participante.

Na Figura 4.5, observa-se que nas três etapas, os exercícios com maior número de tentativas erradas foram o 6 e o 7. No exercício 6, grande parte dos participantes precisaram de 5 tentativas na média para acerta-lo. No entanto, esse exercício apresentou uma grande variação do número de submissões erradas, no qual um usuário realizou apenas uma tentativa errada e outro que efetuou 8 tentativas. O exercício 6 também apresenta o maior *outlier* do estudo, no qual um participante executou 18 tentativas para conseguir acertar o exercício. No exercício 7, o número médio de submissões também foi maior que os demais exercícios, sendo menor apenas que o exercício 6. No entanto, apenas 2 participantes da terceira etapa realizam esse exercício, assim esse gráfico mostrou mais os dados da primeira e segunda etapa.

Nos exercícios 3, 4 e 5 os participantes não apresentaram dificuldades, sendo que a maioria das submissões erradas estão entre 1 e 2 tentativas. Novalmente, a maioria dos participantes da terceira etapa não realizou esses exercícios, assim só observamos as tentativas dos participantes que não utilizaram as dicas do sistema.

4.2.3. Dicas Utilizadas

Foi realizado um mapeamento das dicas utilizadas. Conforme apresentado na Figura 4.6, no primeiro exercício existe uma ampla utilização das dicas em comparação aos demais exercícios.

Pode-se considerar que as dicas utilizadas no primeiro exercício já tenham abordado assuntos que poderiam contribuir com os participantes nos demais exercícios, dado ao fato que as dicas que foram criadas apresentavam informações básicas de programação: “Não se esqueça

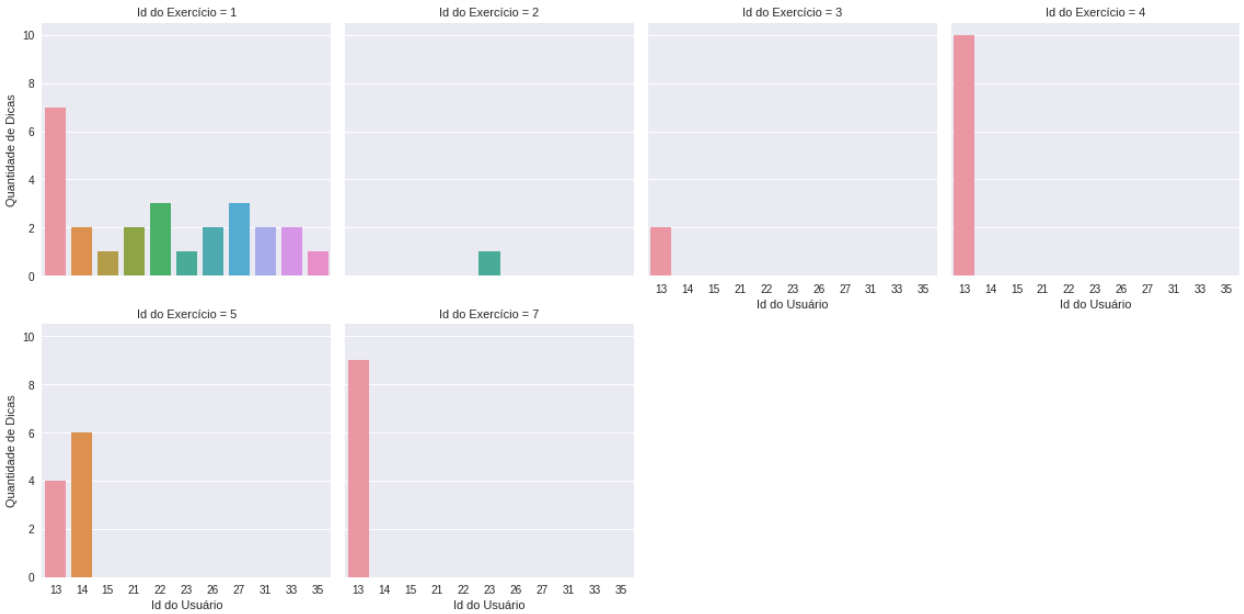

Figura 4.6. Dicas Utilizadas pelos Usuários.

de utilizar precedências entre operadores aritméticos durante as operações matemáticas". Elas também apresentam lembretes importantes sobre a utilização do sistema, como: "Não esqueça de imprimir a saída conforme exemplificado", "Converta todas as entradas para inteiro". Essas dicas podem ser aplicadas em todos os exercícios, tendo em vista que todos os exercícios passaram pela mesma forma de correção e são exercícios básicos que envolvem contas, utilização de condicional e laço de repetição.

Considerando que o primeiro exercício teve um uso expressivo de dicas e que todos os participantes da terceira etapa o realizaram, na Figura 4.7 é mostrado o *boxplot* da quantidade de dicas utilizadas pelos participantes para esse exercício. Percebe-se que a mediana de dicas utilizadas por usuário é 2, variando-se entre a utilização de 1 a 3 dicas. Porém, temos um *outlier* que utilizou 7 dicas para realizar o primeiro exercício.

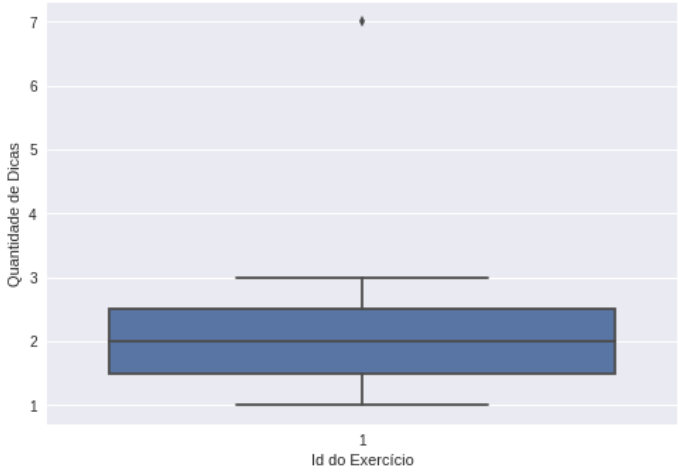

Figura 4.7. Dicas Utilizadas no Primeiro Exercício.

Finalmente, foi realizado uma análise para classificar as dicas conforme a utilização e avaliação produzidas pelos usuários. Cada participante pode consultar quantas dicas acreditasse ser necessárias para realizar as submissões e avaliá-la após o uso, em uma escala de 1 a 5, sendo 1 uma dica de pouca utilidade e 5 uma dica excelente. Na Figura 4.8, é possível observar os indícios do efeito que as dicas tiveram sobre os usuários. Desconsideramos os exercícios 2, 5 e 6 por não conterem dados suficientes para análise. Percebe-se o efeito da qualidade das dicas no primeiro exercício, o qual teve a maior utilização de dicas, conforme discutido anteriormente. Nele foram utilizadas 11 dicas, sendo 4,5 o valor da mediana dessas dicas. Com isso, pode-se supor que a utilização das dicas ajudaram os participantes na realização dos exercícios. O resultado é similar quanto às dicas para os exercícios 3 e 7. Observa-se que, para o exercício 3, a quantidade de submissões erradas na terceira etapa foi baixa, o que pode ser uma consequência da qualidade das dicas.

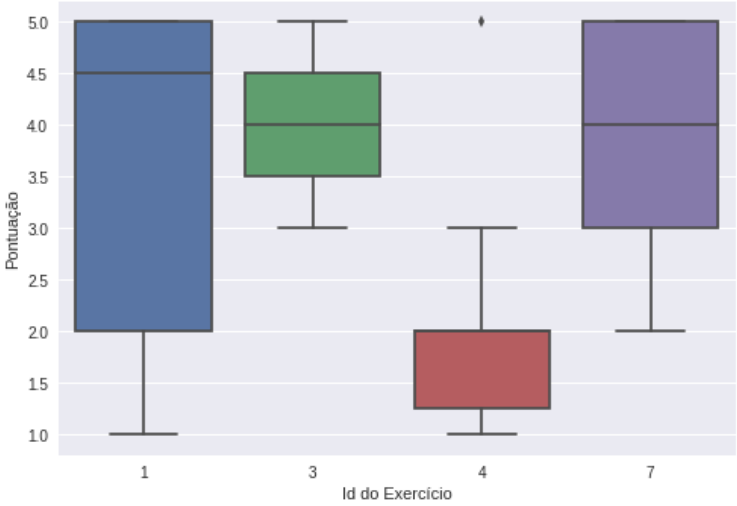

Figura 4.8. Avaliação das Dicas por Exercício

Por outro lado, o exercício 7 foi realizado por apenas dois usuários e as dicas tiveram boas avaliações, mas não é possível verificar se as dicas realmente ajudaram os participantes por causa da falta de dados. No sétimo exercício o usuário 13 realizou 9 tentativas erradas e a notas das dicas são altas, com isso não se pode afirmar que as dicas ajudaram de fato o usuário no sétimo exercício.

4.2.4. Medidas do Código

O sistema realizou a extração das medidas de linhas de código e complexidade ciclomática dos exercícios realizados. Com isso, foi realizado uma média de cada medida, sendo a média de linhas de código e complexidade ciclomática separada por exercício para cada usuário das etapas. Em todos os exercícios, a média das linhas permaneceu quase iguais para cada usuário, assim não foi possível supor que a utilização das dicas ajudou nessa medida. Da mesma forma, a complexidade

ciclomática média dos exercícios realizados pelos participantes, se manteve quase no mesmo valor em todos os exercícios. Nessa análise, pode-se constatar que os exercícios escolhidos não possuiam muitas formas de serem realizados, por serem exercícios básicos de programação. Assim, as médias nas duas medidas apresentaram os mesmos padrões de valores.

4.2.5. Questionário

Aplicou-se o questionário de avaliação do sistema de dicas ao final da terceira etapa, para capturar as opiniões que os participantes tiveram ao utilizar o sistema. A Tabela 4.4 apresenta o preenchimento realizado por dois participantes. Infelizmente, os outros participantes da terceira etapa não preencheram o questionário. Todavia, foram esses dois usuários que realizaram todos os exercícios da terceira etapa.

Tabela 4.4. Questionário Aplicado

	Respostas	
Perguntas	Usuário 13	Usuário 14
Foi fácil utilizar o Sistema de Dicas?	Sim	Sim
Precisou do apoio de outra pessoa para utilizar o sistema?	Sim	Não
O sistema oferece todas as informações necessárias para completar as tarefas?	Sim	Não
O mecanismo de dicas ajudou na resolução de exercícios?	Sim	Sim
Foi fácil acessar as dicas pelo sistema?	Sim	Sim
As dicas foram apresentadas de forma clara e objetiva?	Sim	Sim
Alguma observação sobre o sistema ou o estudo?	Legal , as dicas ajudam de mais.	Talvez poderia deixar as dicas sempre visíveis

O questionário preenchido mostra que, para os dois usuários, a experiência da utilização do sistema de dicas foi positiva, ressaltando que o usuário 14 teve que pedir auxílio ao aplicador do estudo. No entanto, nas perguntas referente ao mecanismo de dicas, os dois sinalizaram que as dicas foram úteis para o desenvolvimento dos exercícios.

4.3. Ameaças a Validade

Uma importante ameaça a validade do nosso estudo foi a baixa quantidade de participantes. Não foi possível aplicar testes estatísticos para aprofundar as análises dos dados. Outra limitação foi a dificuldade dos alunos em escrever programas que pudessem sem executados em linha de comando,

com passagem de parâmetros (conforme requerido pelo avaliador automático). Na realização dos estudos, alguns dos participantes alegaram que nunca tinham realizado a compilação dos exercícios pela linha de comando, que apenas utilizavam IDEs para realizar a compilação.

Capítulo 5

Conclusões

A área de sistema que auxiliam no aprendizado de alunos é bem ativa na comunidade científica. Este trabalho executou a criação de um sistema de dicas, sendo desenvolvido utilizando o *framework* Laravel. O sistema de dicas possibilita: gerenciamento de, usuários, exercícios, lista de exercícios, dicas, realização de submissões de exercícios e correções automatizadas dos exercícios submetidos.

A maioria dos trabalhos relacionados a dicas emprega técnicas de inteligência artificial para realizar a criação de dicas automáticas, eliminando a necessidade do acompanhamento presencial dos professores quando os alunos estão realizando os exercícios. Este trabalho pretendeu explorar melhor o aprendizado do aluno, possibilitando melhorar suas habilidades de programação resolvendo exercícios e criando as dicas. Desta forma, a própria criação de dicas, além da resolução dos exercícios, constitui uma forma de aprendizagem. O sistema implementado permite a condução de estudos quanto ao uso de dicas de forma simples e também em multidão e, de certa forma, fomentando o aprendizado colaborativo e ativo.

Para realizar a avaliação do sistema, foi analisado os dados criados a partir das etapas do estudo. Apesar que poucos usuários utilizaram o sistema, as dicas criadas foram bem avaliadas e os resultados do questionário, aplicado ao final da terceira etapa, indicou que as dicas foram úteis para a resolução dos exercícios.

Dado o desenvolvimento deste trabalho, pode-se apontar alguns trabalhos futuros necessários para melhorar a utilização do sistema de dicas. Dado que houve poucos participantes no estudo, seria preciso realizar mais etapas do estudo para aumentar a quantidade de dados e aplicar métodos estatísticos para avaliar o sistema. Além disso, poderia ser aplicado o *leanersoucing* no fluxo de execução do sistema, semelhando o apresentado por Glassman et al. (2016). Nele seria possível verificar se o mecanismo de dicas realmente melhora as habilidades em programação dos alunos.

Uma melhoria com relação ao sistema seria a criação de vários tipos de dicas, a fim de abranger aspectos globais e específicos da programação. Também se estabelece como trabalho

futuro a criação de um modelo de recomendação de dicas, para que o sistema possa recomendar dicas de acordo com as dificuldades especificas dos alunos. Também são necessárias melhorias no *layout* da aplicação, para facilitar a leitura e utilização das dicas, e a criação de um *dashboard* melhor para o administrador, em que seria possível acompanhar as estatísticas em tempo real dos dados gerados pelos usuários durante a realização dos exercícios.

Referências

ANTONUCCI, Paolo; ESTLER, Christian; NIKOLIć, Durica; PICCIONI, Marco; MEYER, Bertrand. An incremental hint system for automated programming assignments. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2015. (ITiCSE '15), p. 320–325. ISBN 978-1-4503-3440-2.

BOSCH, Jan; MOLIN, Peter; MATTSSON, Michael; BENGTSSON, PerOlof. Object-oriented framework-based software development: Problems and experiences. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 32, n. 1es, mar. 2000. ISSN 0360-0300.

BOSSE, Yorah; GEROSA, Marco Aurélio. Reprovações e trancamentos nas disciplinas de introdução à programação da universidade de são paulo: Um estudo preliminar. In: *XXIII WEI–Workshop sobre Educação em Informática. Recife, Julho*. Recife, PB, Brasil: WEI, 2015.

CUKIERMAN, Diana. Predicting success in university first year computing science courses: The role of student participation in reflective learning activities and in i-clicker activities. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2015. (ITiCSE '15), p. 248–253. ISBN 978-1-4503-3440-2.

CUMMINS, Stephen; STEAD, Alistair; JARDINE-WRIGHT, Lisa; DAVIES, Ian; BERESFORD, Alastair R.; RICE, Andrew. Investigating the use of hints in online problem solving. In: *Proceedings of the Third (2016) ACM Conference on Learning @ Scale*. New York, NY, USA: ACM, 2016. (L@S '16), p. 105–108. ISBN 978-1-4503-3726-7.

ELKHERJ, Matthew; FREUND, Yoav. A system for sending the right hint at the right time. In: *Proceedings of the First ACM Conference on Learning @ Scale Conference*. New York, NY, USA: ACM, 2014. (L@S '14), p. 219–220. ISBN 978-1-4503-2669-8.

EVANS, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Atlanta, GA, USA: Addison-Wesley, 2004.

FAYAD, Mohamed; SCHMIDT, Douglas C. Object-oriented application frameworks. *Communications of the ACM*, ACM, v. 40, n. 10, p. 32–38, 1997.

GLASSMAN, Elena L.; LIN, Aaron; CAI, Carrie J.; MILLER, Robert C. Learnersourcing personalized hints. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. New York, NY, USA: ACM, 2016. (CSCW '16), p. 1626–1636. ISBN 978-1-4503-3592-8.

LAHTINEN, Essi; ALA-MUTKA, Kirsti; JÄRVINEN, Hannu-Matti. A study of the difficulties of novice programmers. In: ACM. *ACM SIGCSE Bulletin*. New York, NY, USA, 2005. v. 37, n. 3, p. 14–18.

LITTLEFAIR, Tim. *CCCC - C and C++ Code Counter*. 1998. Disponível em: <http://cccc.sourceforge.net/>.

MCCABE, Thomas J. A complexity measure. In: *Proceedings of the 2Nd International Conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976. (ICSE '76), p. 407–.

PAQUETTE, Luc; LEBEAU, Jean-François; BEAULIEU, Gabriel; MAYERS, André". Automating next-step hints generation using astus. In: ____. *Intelligent Tutoring Systems: 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 201–211. ISBN 978-3-642-30950-2.

PHPGROUP. *PHP 7.2.0RC6 Released*. 2017. Disponível em: <https://secure.php.net/archive/2017.php\#id2017-11-09-1>.

PHPGROUP. *What is PHP?* 2017. Disponível em: <https://secure.php.net/manual/en/intro-whatis.php>.

PRICE, Thomas W. Integrating intelligent feedback into block programming environments. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. New York, NY, USA: ACM, 2015. (ICER '15), p. 275–276. ISBN 978-1-4503-3630-7.

RAZZAQ, Leena; HEFFERNAN, Neil T. Hints: Is it better to give or wait to be asked? In: ____. *Intelligent Tutoring Systems: 10th International Conference, ITS 2010, Pittsburgh, PA, USA, June 14-18, 2010, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 349–358. ISBN 978-3-642-13388-6.

REENSKAUG, Trygve. Models-views-controllers. *Technical note, Xerox PARC*, Palo Alto, CA, USA, v. 32, n. 55, p. 6–2, 1979.

Apêndices

Apêndice A

Termo de Consentimento

Este documento visa solicitar sua autorização para participar do experimento do estudo sobre o uso do mecanismo de dicas no ensino de conceitos básicos de programação. Informamos que esse estudo se vincula ao trabalho do aluno da UTFPR Gustavo Correia Gonzalez para a disciplina de Trabalho de Conclusão de Curso 2. Os dados coletados não serão repassados de forma individualizada a terceiros e não serão usados para outros fins. Não haverá prejuízo acadêmico de nenhuma natureza por informações disponibilizadas para esta pesquisa. Por intermédio deste Termo lhe é garantido os seguintes direitos:

- solicitar maiores esclarecimentos sobre esta pesquisa;
- sigilo absoluto sobre nomes ou quaisquer outras informações que possam levar à identificação pessoal;
- ampla possibilidade de negar-se a responder a quaisquer questões ou a fornecer informações que julguem prejudiciais à sua integridade física, moral e social;
- opção de solicitar que determinadas falas e/ou declarações não sejam incluídas em nenhum documento, o que será prontamente atendido;
- desistir, a qualquer tempo, de participar da Pesquisa.

Pessoas que acompanharão o estudo e terão acesso ao material:

- Aluno da UTFPR: Gustavo Correia Gonzalez.
- Professor da UTFPR: Marco Aurélio Graciotto Silva.
- Professor da UTFPR: Igor Wiese.

Apêndice

B

Questões Utilizadas no Sistema

Questão 1: Joãozinho quer calcular e mostrar a quantidade de litros de combustível gastos em uma viagem, ao utilizar um automóvel que faz 12 KM/L. Para isso, ele gostaria que você o auxiliasse através de um simples programa. Para efetuar o cálculo, deve-se fornecer o tempo gasto na viagem (em horas) e a velocidade média durante a mesma (em km/h). Assim, pode-se obter distância percorrida e, em seguida, calcular quantos litros seriam necessários. Mostre o valor com 3 casas decimais após o ponto.

Questão 2: Leia quatro valores inteiros A, B, C e D. A seguir, calcule e mostre a diferença do produto de A e B pelo produto de C e D segundo a fórmula: DIFERENCA = (A * B - C * D).

Questão 3: Leia quatro valores inteiros A, B, C e D. A seguir, calcule e mostre a diferença do produto de A e B pelo produto de C e D segundo a fórmula: DIFERENCA = (A * B - C * D).

Questão 4: Leia um valor inteiro correspondente há idade de uma pessoa em dias e informe-a em anos, meses e dias. Obs.: apenas para facilitar o cálculo, considere todo ano com 365 dias e todo mês com 30 dias. Nos casos de teste nunca haverá uma situação que permite 12 meses e alguns dias, como 360, 363 ou 364. Este é apenas um exercício com objetivo de testar raciocínio matemático simples.

Questão 5: Leia um valor inteiro, que é o tempo de duração em segundos de um determinado evento em uma fábrica, e informe-o expresso no formato horas:minutos:segundos.

Questão 6: Leia 3 valores de ponto flutuante e efetue o cálculo das raízes da equação de Bhaskara. Se não for possível calcular as raízes, mostre a mensagem correspondente "Impossivel calcular", caso haja uma divisão por 0 ou raiz de número negativo.

Questão 7: Faça um programa que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o total a receber no final do mês, com duas casas decimais.