UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MAURÍCIO FERRON CARNEIRO

SISTEMA WEB PARA AUXÍLIO NO TREINAMENTO PARA MARATONAS DE PROGRAMAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2012

MAURÍCIO FERRON CARNEIRO

SISTEMA WEB PARA AUXÍLIO NO TREINAMENTO PARA MARATONAS DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz T. Borsoi

PATO BRANCO
2012

ATA Nº: **197**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **MAURíCIO FERRON CARNEIRO**.

Às 10:30 hrs do dia 9 de outubro de 2012, Bloco S da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Omero Francisco Bertol (Convidado) e Géri Natalino Dutra (Convidado), para avaliar o Trabalho de Diplomação do aluno Maurício Ferron Carneiro, matrícula 910023, sob o título **Sistema Web para Auxílio no Treinamento para Maratonas de Programação**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 11:10 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Omero Francisco Bertol, M.Sc.
Convidado

Prof. Géri Natalino Dutra, M.Sc.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

CARNEIRO, Maurício Ferron. Sistema web para auxílio no treinamento para maratonas de programação. 2012. 59 f. Relatório de Estágio Supervisionado - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2012.

Maratonas e olimpíadas, dentre outras denominações, de programação geralmente visam testar a capacidade e a rapidez de raciocínio lógico dos participantes na resolução de problemas computacionais. O treino, baseado na resolução de problemas, pode auxiliar a aprimorar o raciocínio lógico e a agilidade de resolução. Desta forma, verificou-se a oportunidade de desenvolver um sistema *web* para disponibilizar problemas, materiais de apoio, problemas resolvidos, dicas e orientações. As listas de exercícios podem ser compostas a partir de problemas cadastrados. A esses problemas podem estar associadas dicas de resolução. O sistema também permitirá a disponibilização de funções implementadas em código. A linguagem escolhida foi a PHP com a implementação do padrão *Model-View-Controller*. Esse padrão foi escolhido pela facilidade que o mesmo provê de separação entre interface, lógica de negócio e acesso a dados.

Palavras-chave: Model-View-Controller. PHP. Maratonas de programação. Sistema Web.

ABSTRACT

CARNEIRO, Maurício Ferron. Web system to aid in trainning for programming contests. 2012. 59 f. Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2012.

Programming contests, such as marathons and Olympics of programming, generally aim to test the ability of logical reasoning and speed of participants in solving computational problems. The training, based on problem solving, can help to improve the logical reasoning and speed of resolution. Thus, there was an opportunity to develop a system to provide web problems, support materials, solved problems, tips and guidelines. Lists of exercises can be composed from problems registered. The problems may be associated with troubleshooting tips. The system also allows the provision of functions implemented in code. The language used to implement the system is PHP with the implementation of Model-View-Controller. This pattern was chosen by the facilities provided, such as separation between interface, business logic and data access.

Palavras-chave: Model-View-Controller. PHP. Programming contests. Web system.

LISTA DE FIGURAS

Figura 1 – Padrão arquitetural MVC 17
Figura 2 – Ferramenta de modelagem Astah..... 24
Figura 3 – IDE NetBeans 25
Figura 4 – Diagrama de casos de uso..... 33
Figura 5 – Diagrama de entidades e relacionamentos do banco de dados 38
Figura 6 – Tela de login ao sistema 39
Figura 7 – Tela inicial do sistema..... 39
Figura 8 – Tela de níveis do sistema..... 40
Figura 9 – Tela inclusão de um nível..... 40
Figura 10 – Tela com a listagem das questões 41
Figura 11 – Tela para cadastro de uma nova questão 41
Figura 12 – Tela para alterar uma questão 42
Figura 13 – Tela inclusão de um nível..... 43

LISTA DE QUADROS

Quadro 1 – Iterações definidas 28
Quadro 2 – Caso de uso baixar arquivo de entrada 34
Quadro 3 – Caso de uso verificar respostas 34
Quadro 4 – Caso de uso consultar resposta 35
Quadro 5 – Caso de uso pesquisar por listas 35
Quadro 6 – Caso de uso consultar material de apoio 36
Quadro 7 – Caso de uso compor listas 36
Quadro 8 – Caso de uso manter questões 36
Quadro 9 – Caso de uso manter material de apoio 37
Quadro 10 – Caso de uso manter rotinas 37
Quadro 11 – Caso de uso manter dicas 38

LISTAGENS DE CÓDIGO

Listagem 1 – Validação de acesso do administrador 44
Listagem 2 – Template do menu 45
Listagem 3 – Função para proteger a página 45
Listagem 4 – Função para redirecionar página em login incorreto 46
Listagem 5 – Função AdminController.php() 47
Listagem 6 – Função sql.admin.class() 48
Listagem 7 – Função alteraNiveisController.php() 49
Listagem 8 – Controle das questões 52
Listagem 9 – Funções para operações no banco de dados 53
Listagem 10 – Comparação de arquivos de resposta da questão 55

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
ACM	*Association for Computing Machinery*
API	*Application Programming Interface*
HTML	*HyperText Markup Language*
ICPC	*International Collegiate Programming Contest*
IDE	*Integrated Development Environment*
IMAP	*Internet Message Application Protocol*
IOI	*International Olympiad in Informatics*
MVC	*Model-View-Controller*
OBI	Olimpíada Brasileira de Informática
PHP	*PreProcessor Hipertext PHP*
POP3	*Post Office Protocol*
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO 10
1.1 CONSIDERAÇÕES INICIAIS 10
1.2 OBJETIVOS 10
1.2.1 Objetivo Geral 11
1.2.2 Objetivos Específicos 11
1.3 JUSTIFICATIVA 11
1.4 ESTRUTURA DO TRABALHO 12
2 O PADRÃO MVC NO DESENVOLVIMENTO DE APLICAÇÕES WEB 13
2.1 APLICAÇÕES WEB 13
2.2 PADRÕES E ESTILOS ARQUITETURAIS 14
2.2.1 Padrão MVC 15
3 MARATONAS DE PROGRAMAÇÃO E OLIMPÍADAS DE INFORMÁTICA 19
3.1 FORMA DE FUNCIONAMENTO DE UMA COMPETIÇÃO 19
3.2 ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST 19
3.3 MARATONA BRASILEIRA DE PROGRAMAÇÃO 20
3.4 OLIMPÍADA INTERNACIONAL DE INFORMÁTICA 20
3.5 OLIMPÍADA BRASILEIRA DE INFORMÁTICA 21
4 MATERIAIS E MÉTODO 23
4.1 MATERIAIS 23
4.1.1 Astah Community 23
4.1.2 NetBeans 25
4.1.3 Linguagem PHP 26
4.1.4 phpMyAdmin 26
4.1.5 MySQL 26
4.1.6 Tomcat 27
4.2 MÉTODO 27
5 RESULTADOS E DISCUSSÃO 30
5.1 APRESENTAÇÃO DO SISTEMA 30
5.2 MODELAGEM DO SISTEMA 30
5.3 DESCRIÇÃO DO SISTEMA 39
5.4 IMPLEMENTAÇÃO DO SISTEMA 42
6 CONCLUSÃO 56
REFERÊNCIAS 57

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, o objetivo e a justificativa deste trabalho. O capítulo é finalizado com a organização do texto por meio da apresentação dos capítulos que o compõe.

1.1 CONSIDERAÇÕES INICIAIS

As primeiras aplicações *web* desenvolvidas utilizando HTML (*HyperText Markup Language*) geralmente mesclam o código escrito em uma linguagem de programação como PHP (*PreProcessor Hipertext PHP*), por exemplo, com as *tags* de formatação do texto que define a interface do aplicativo e as instruções (sentenças) SQL (*Structured Query Language*) para manipulação dos dados. Essa forma de programação dificulta o reuso de código, a manutenção e mesmo o entendimento do programa, especialmente por outros programadores.

A programação orientada a objetos foi uma forma de promover o reuso de código, mesmo que seja pelo mecanismo de herança. Contudo, não propiciou uma solução efetiva para a mescla de código e formatação em aplicativos *web* baseados em HTML, por exemplo.

Uma forma mais efetiva de promover a separação do código ocorre pelos padrões de projetos, a exemplo do *Model-View-Controller* (MVC). O padrão MVC provê uma separação entre a visão (a interface com o usuário), a camada que contém os dados e a camada que contém a lógica de negócio. Com o uso deste padrão uma mesma aplicação (dados e lógica de negócio) pode ter diferentes visões: ser acessada por interfaces distintas ou ter os mesmos dados apresentados de formas distintas.

Como forma de exemplificar a aplicação do padrão MVC este trabalho apresenta a implementação do padrão MVC utilizando a linguagem PHP. É um sistema para treinamento de maratonas de programação. Com o uso de MVC uma versão para dispositivos móveis, por exemplo, poderá ser mais facilmente implementada.

1.2 OBJETIVOS

O objetivo geral se refere ao principal resultado obtido com o desenvolvimento deste trabalho e os objetivos específicos o complementam.

1.2.1 Objetivo Geral

Implementar um sistema *web* que visa auxiliar no preparo de alunos para realização de maratonas de programação.

1.2.2 Objetivos Específicos

Os objetivos específicos listados a seguir complementam o objetivo geral do trabalho:

- Apresentar a implementação do padrão de projetos MVC utilizando a linguagem PHP.
- Fornecer uma forma facilitada para que alunos possam obter material para estudo e treinamento para maratonas de programação, bem como algoritmos computacionais.

1.3 JUSTIFICATIVA

O desenvolvimento de um aplicativo *web* para treinamento para maratonas de programação, especificamente a Maratona Brasileira de Programação, que é uma fase da Maratona Mundial de Programação, facilita o acesso dos usuários. Os alunos podem acessar exercícios e material auxiliar de qualquer computador com acesso à Internet, desde que possuam um navegador *web* instalado.

O sistema também possibilitará que seja submetido um arquivo com a saída dos dados gerados pelo programa e esse arquivo é comparado com a saída que está armazenada no banco de dados do sistema e relacionada ao referido problema. Assim, o usuário saberá se a solução apresentada está correta. O sistema desenvolvido como resultado deste trabalho verifica apenas a saída gerada que é um arquivo no formato TXT (abreviação de TEXT, ou texto em inglês).

O uso do padrão de projetos MVC permite segmentar o aplicativo em camadas de forma a separar a interface, do modelo de negócio e do tratamento dos dados. Assim, futuramente poderá ser implementada uma versão para dispositivos móveis, por exemplo, em que apenas a visualização (interface) é alterada. O modelo de dados e as regras de negócio permanecem inalterados.

1.4 ESTRUTURA DO TRABALHO

O Capítulo 1 apresenta as considerações iniciais, os objetivos e a justificativa.

No Capítulo 2 está o referencial teórico que se refere aos padrões de projeto, centrado em MVC.

No Capítulo 3 é apresentado sobre maratonas e olimpíadas de programação visando expor o contexto da aplicação desenvolvida. O conteúdo desse capítulo foi baseado em informações obtidas da Internet, especialmente de páginas *wiki*.

No Capítulo 4 são apresentadas as tecnologias e as ferramentas utilizadas no desenvolvimento do trabalho e o método utilizado.

O Capítulo 5 contém o resultado da realização deste trabalho, com a modelagem do sistema e a exemplificação da implementação do padrão MVC.

Já no Capítulo 6 está a conclusão com as considerações finais.

Finalizando com as referências bibliográficas que fundamentaram este trabalho de conclusão de curso.

2 O PADRÃO MVC NO DESENVOLVIMENTO DE APLICAÇÕES WEB

Este capítulo contém o referencial teórico do trabalho. Inicialmente é apresentado o contexto de aplicação dos padrões arquiteturais. A Seção 2.2 se refere à arquitetura de *software*, com uma visão geral e conceitual do assunto em que padrões arquiteturais de projeto se inserem e em seguida é tratado especificamente sobre o padrão arquitetural MVC. Esse padrão é utilizado na implementação do sistema objeto deste trabalho.

2.1 APLICAÇÕES WEB

De maneira geral, uma aplicação *web* é aquela que é executada utilizando recursos da Internet e é acessada por meio de um navegador *web.* O navegador permite a execução da interface de interação da aplicação. A ideia básica das aplicações *web* é a de uma aplicação do tipo cliente/servidor em que há um servidor e clientes, ambos *web*, que acessam a aplicação. E clientes e servidor são *web*.

A arquitetura cliente/servidor em uma aplicação *web*, em termos de serviços, está organizada da seguinte forma: de um lado está o cliente, navegador ou *browser web*, que solicita dados ao servidor, recebe as respostas enviadas pelo servidor, formata os dados e os apresenta ao usuário; do outro lado está o servidor que recebe as requisições dos clientes, processa essas requisições e se necessário lê dados armazenados em arquivos e retorna dados para o cliente.

O padrão para desenvolvimento de páginas *web* é HTML. HTML é linguagem de marcação baseada em *tags* que definem o formato do contexto contido nessas *tags,* que são especificadas por padrão pela linguagem. Nas aplicações que possuem a interface definidas como páginas HTML, o código desenvolvido com a linguagem de programação para implementar a lógica de negócio e a manipulação de dados fica junto com essas *tags* HTML. Nesse modelo, as *tags* que formatam o conteúdo da página, sejam campos de formulário ou dados que comporão uma tabela, ficam mesclados com o código. Isso dificulta a implementação, o entendimento e a manutenção da aplicação.

Assim, equipes distintas, por exemplo, a equipe que define a interface da aplicação e a equipe que implementa o código não podem trabalhar simultaneamente em um mesmo formulário. A manutenção é dificultada porque fica mais difícil localizar erros ou mesmo incluir alterações ou incrementos se o código escrito na linguagem de programação utilizada

está junto com a HTML da página. E o entendimento do código é dificultado pela junção entre instruções de formatação e da linguagem de programação. Outra dificuldade está relacionada ao reuso. Se o código estivesse separado da interface, um mesmo código de um determinado cadastro poderia ser reusado independentemente da interface definida para o mesmo.

Uma das maneiras de minimizar esse problema de alto acoplamento entre a camada de interface, de lógica de negócio e de banco de dados ocorre por meio do uso do MVC. O padrão de projeto (*design pattern*) MVC possibilita desvincular a manipulação de dados persistentes, da lógica de negócio e da lógica de visualização.

O padrão de projeto MVC é um padrão de projeto arquitetural. Esse tipo de padrão define os elementos, as relações e as regras a serem seguidas e que já tiveram sua utilidade avaliada na solução de problemas (GERMOGLIO, 2010).

2.2 PADRÕES E ESTILOS ARQUITETURAIS

Um padrão pode ser considerado como uma solução de projeto, que pode ser reusada em problemas semelhantes. Um padrão de projeto ou padrão arquitetural define elementos, relações entre elementos e regras a serem seguidas.

Germoglio (2010) não considera padrão de projeto e padrão arquitetural como sinônimos. Para esse autor, a principal diferença entre um padrão arquitetural, também denominado de estilo arquitetural, e um padrão de projeto, também denominado *design pattern*, é que padrão arquitetural se refere aos problemas em nível arquitetural, tornando-se mais abrangente e padrão de projeto tem efeito mais restrito, relacionado ao *software*. Assim, é comum que os padrões de projeto estejam associados à implementação do sistema, ao uso de tecnologias. E os padrões arquiteturais estejam mais voltados para a solução do problema, em um escopo de mais alto nível.

Para Buschmann et al. (1996), padrões arquiteturais expressam um esquema de organização estrutural fundamental para sistemas de *software*, definindo seus subsistemas, especificando suas responsabilidades e incluindo regras e orientações para a organização dos relacionamentos entre os subsistemas.

Gamma et al. (2000) definem que padrões de projeto provêem um esquema para o refinamento dos subsistemas (ou componentes) de um sistema de *software* e dos relacionamentos entre eles. Descrevem uma estrutura recorrente de objetos comunicantes que

resolve um problema de projeto genérico dentro de um contexto particular (GAMMA et al., 2000).

Buschmann et al. (1996), acrescentam o conceito de padrões de codificação (ou idiomas). Para esses autores padrões de codificação são padrões de baixo nível de abstração, específicos para uma linguagem de programação particular. Descrevem como implementar aspectos particulares de componentes ou dos relacionamentos entre eles, usando as características de uma dada linguagem de programação.

Contudo, é possível encontrar padrões inicialmente descritos como arquiteturais sendo utilizados mais especificamente na solução tecnológica do problema e vice-versa. Desta forma, neste texto padrões de projeto e arquitetural são utilizados indistintamente.

McConnell (2004) ressalta como benefícios do uso de padrões em um projeto os seguintes:

a) Redução da complexidade da solução - um padrão arquitetural define elementos e relações entre esses elementos, diminuindo a quantidade de novos conceitos a serem utilizados na solução.

b) Reuso - como o padrão de projetos representa a solução de um problema recorrente, a implementação (parcial ou total) do padrão é reusada em contextos semelhantes.

c) Geração de alternativas - padrões arquiteturais distintos podem resolver um mesmo problema. Isso permite a escolha entre soluções possíveis, considerando os benefícios e as desvantagens de cada um delas.

d) Facilidade de comunicação - os padrões arquiteturais descrevem conceitos e elementos que estarão presentes no projeto. Isso facilita a comunicação entre os envolvidos na implementação do projeto de *software*.

Buschmann et al. (1996) descrevem 23 padrões de projeto, dentre os quais estão: *layers* (camadas), *pipes* e *filters*, *Model-View-Controller e microkernel*. Alguns dos padrões expostos em Buschmann et al. (1996) foram propostos por Garlan e Shaw (1994). O padrão MVC é assunto da próxima subseção por ser o utilizado no desenvolvimento do sistema que é resultado da realização deste trabalho.

2.2.1 Padrão MVC

O Model-View-Controller é um padrão de projeto frequentemente utilizado em aplicações que necessitam manter múltiplas visões de um mesmo conjunto de dados. O MVC

provê uma separação clara de objetos em três partes: modelo, visão e controle. Por meio dessa separação, múltiplas visões e controles podem interagir com um mesmo modelo de dados. E novas visões e controles podem interagir com o modelo sem necessidade de alterá-lo.

O MVC é um conceito de desenvolvimento e projeto que visa organizar a separação de uma aplicação em três partes distintas: modelo, visão e controle (GAMMA et al., 2000; GONÇALVES et al., 2005; SWEAT, 2005; HANSEN, FOSSUM, 2005):

a) Modelo (*model*) - o modelo encapsula os dados do aplicativo, bem como os métodos para acessar e manipular esses dados. O modelo é conhecido como camada de dados. A camada de dados consiste nos componentes que representam e armazenam a informação exibida nessas páginas. O modelo de dados é composto por objetos ou elementos que reproduzem entidades e ações do mundo real ou tabelas de uma base de dados. O modelo é uma camada passiva que não possui qualquer funcionalidade além do armazenamento de dados. Desta forma, um modelo não deve conter funcionalidades relacionadas com a interface com o utilizador.

O modelo é responsável por manter o estado da aplicação. Ele é mais do que uma classe para armazenar os dados, nele devem estar contidas todas as regras de negócio que se aplicam a esses dados. O modelo também é responsável por comunicar-se com o banco de dados, se for necessário.

b) Visão (*view*) - a visão está relacionada a exibir os dados ou informações da aplicação para o usuário e por isso é conhecida como camada de apresentação. A visão extrai dados do modelo e os formata para apresentação. A camada de apresentação consiste dos componentes que são visíveis para o utilizador da aplicação, como, por exemplo, as páginas *web* que são exibidas no navegador. A visão especifica exatamente como o modelo deve ser apresentado. É a interface do usuário. A visão é dinâmica, se adequando a qualquer modificação no modelo.

c) Controle (*control*) – o controle representa o processo de negócio. É responsável pelo processamento de dados e atualiza o modelo e a visão apropriadamente. O controle direciona o fluxo do aplicativo e recebe entradas, traduzindo-as para o modelo e a visualização. A camada de controle é a intermediária entre a camada de visão e de modelo que representa o processo de negócio, ou de outra forma, que é responsável pelo processamento dos dados. O controle conhece a estrutura do modelo e da visão. O controle realiza a pesquisa de dados para apresentação na visão, assim como recebe os dados submetidos pelo utilizador por meio da visão (por formulários, por exemplo) e realiza o seu armazenamento na camada de dados.

O controle traduz as interações do usuário com a visão, mapeando-as para tarefas que o modelo realizará. Em uma aplicação *standalone*, a interação do usuário poderia ser um clique em um botão, por exemplo. Em uma aplicação *web*, pode ser o clique em um *link*, por exemplo.

A Figura 1 apresenta uma visão geral do MVC por meio das suas principais partes (retângulos que representam Visão, Modelo e Controle). Na representação básica no MVC, com suas três partes, foi acrescentada a interação que ocorreria por meio de uma interface *web* (por ser o foco deste trabalho). E, ainda, o vínculo com os dados e as regras de negócio, que definem a aplicação em si.

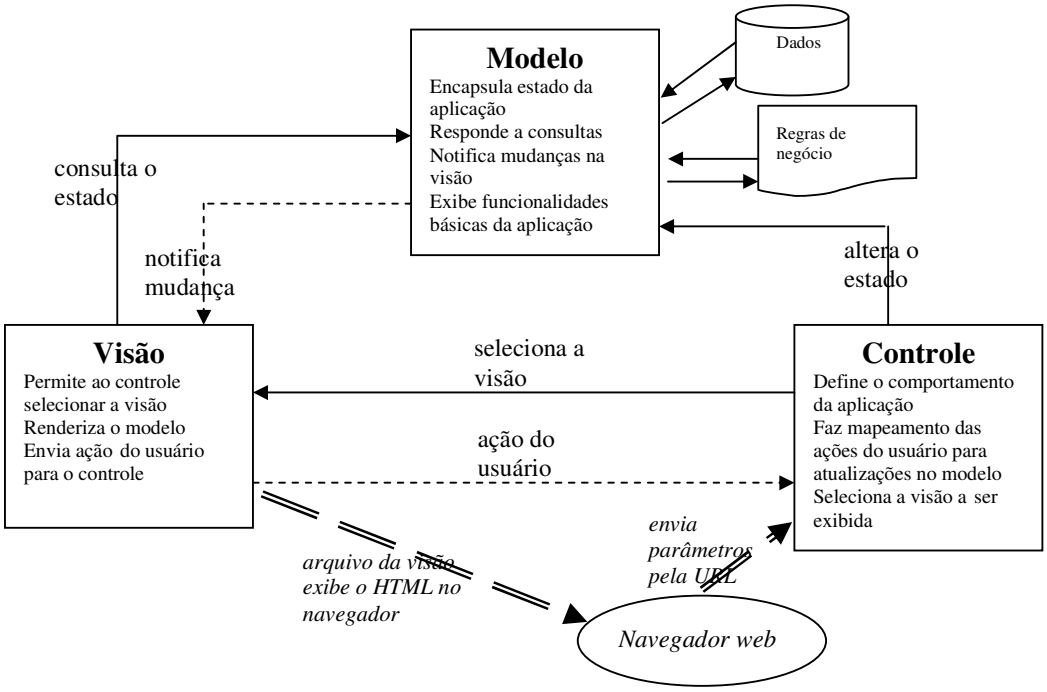

Figura 1 – Padrão arquitetural MVC
Fonte: baseado em Tiwari e Mittal (2012, p.1).

Pela representação da Figura 1, a atuação do MVC inicia com um evento que é recebido pelo controle. O fluxo de dados representado na Figura 1 expressa que a partir de um evento iniciador, o controle altera a visão ou o modelo. A visão por sua vez obtém dados do modelo e o modelo atualiza a visão quando ocorrem mudanças nos dados. Eventos geralmente fazem com que um controle altere um modelo, uma visão ou ambos. Sempre que um controle alterar dados ou propriedades de um modelo, todas as visões dependentes são automaticamente atualizadas.

O MVC tem como principal objetivo os dados e a lógica de negócio (*Model*), o fluxo

da aplicação (*Controller*) e a interface com usuário (*View*). Assim, a lógica de negócios pode ser acessada e visualizada por meio de várias interfaces e para a visão não importa a origem dos dados, mas ela precisa garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os estados do modelo mudam, o modelo notifica as visões para que elas sejam atualizadas.

No desenvolvimento de aplicações *web* o uso do MVC encontra um problema que é o fato que aplicações *web* são intrinsecamente segmentadas entre cliente e servidor. E a visão é apresentada no cliente, mas o modelo e o controle podem ser, teoricamente, segmentados em uma diversidade de maneiras entre cliente e servidor (LEFF; RAYFIELD, 2001). E a forma como o acoplamento entre as três partes desse modelo é realizada tem impacto na facilidade de desenvolvimento, manutenção e reuso do código (HANSEN; FOSSUM, 2005).

3 MARATONAS DE PROGRAMAÇÃO E OLIMPÍADAS DE INFORMÁTICA

Este capítulo apresenta conceitos e explicações sobre maratonas e olimpíadas de programação.

3.1 FORMA DE FUNCIONAMENTO DE UMA COMPETIÇÃO

Uma maratona ou olimpíada de programação é uma forma de competição na qual um programador, ou grupo de programadores, deve apresentar uma solução computacional correta para um dado problema. Geralmente, especialmente em eventos de maior porte como a maratona brasileira de programação ou mesmo a mundial, a correção é realizada por meio de testes. Esses testes são compostos por um arquivo contendo as entradas para o problema e outro arquivo com a saída esperada.

O programa (que implementa o algoritmo com a solução fornecida) deve ler o arquivo que contém as entradas e imprimir o resultado correspondente a essas entradas, que é o arquivo de saída. O arquivo contendo as saídas deve ser idêntico ao arquivo usado para verificação, ou seja, o arquivo que contém a saída esperada.

Além de a saída ser idêntica, as maratonas também podem estabelecer outros critérios. Como, por exemplo, o de respeitar limites de tempo de execução e de quantidade de memória usada. Pode ser oferecida pontuação maior para as equipes que apresentarem a solução em tempo menor e/ou com menor quantidade de memória utilizada.

3.2 ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

A ACM (*Association for Computing Machinery*) *International Collegiate Programming Contest* (ICPC) é uma competição anual e mundial de programação entre instituições de ensino superior. A ICPC é uma competição em equipe composta por três estudantes universitários, que tenham tido no máximo cinco anos de educação universitária antes da competição ou tenham até 23 anos até o final do ano anterior ao ano da final mundial (WIKIPEDIA, 2012).

Durante a competição, as equipes têm 5 horas para resolver entre 8 e 11 problemas. As soluções devem ser submetidas, principalmente, em C, C++ ou Java. Os programas são então testados com casos de teste e, se o programa produzir alguma resposta errada ou não for

suficientemente eficiente, ele não é aceito e a equipe é notificada.

O vencedor é a equipe que resolver mais problemas corretamente e em menos tempo. Se necessário, em caso de empate no número de problemas resolvidos, a classificação das equipes é determinada pela soma dos tempos da submissão correta de cada problema mais 20 minutos para cada submissão rejeitada em um problema que acabe sendo resolvido.

A competição ocorre em etapas. Muitas universidades realizam competições internas, chamadas de seletivas, para definir seus representantes na fase regional. No caso do Brasil existem ainda as sub-regionais, que tem o objetivo de definir as equipes que irão para a fase regional. No Brasil a regional é organizada pela Maratona Brasileira de Programação ou Maratona de Programação, que coordena a fase sub-regional e a fase regional da competição.

3.3 MARATONA BRASILEIRA DE PROGRAMAÇÃO

A Maratona Brasileira de Programação é um evento da Sociedade Brasileira de Computação que existe desde o ano de 1996 (SBC, 2012). A Maratona nasceu das competições regionais classificatórias para as finais mundiais da ACM ICPC.

A Maratona Brasileira de Programação se destina aos alunos de cursos de graduação e início de pós-graduação na área de Computação e afins (Ciência da Computação, Engenharia de Computação, Sistemas de Informação, Matemática, entre outras). A competição promove nos alunos a criatividade, a capacidade de trabalho em equipe, a busca de novas soluções de *software* e a habilidade de resolver problemas sob pressão.

As equipes são compostas por três alunos, que tentarão resolver durante 5 horas o maior número possível dos 8 ou mais problemas que são entregues no início da competição. Estes alunos têm à sua disposição apenas um computador e material impresso (livros, listagens, manuais) para resolver os problemas. O time que conseguir resolver o maior número de problemas (no menor tempo acumulado com as penalidades, caso haja empate) é declarado o vencedor (SBC, 2012).

3.4 OLIMPÍADA INTERNACIONAL DE INFORMÁTICA

A Olimpíada Internacional de Informática (*International Olympiad in Informatics* - IOI) é uma das Olimpíadas Internacionais de Ciências. Ela é realizada anualmente desde 1989

e é destinada a alunos do Ensino Médio ou que o tenham cursado no ano anterior.

A competição consiste de problemas computacionais de natureza algorítmica. Os estudantes, sendo quatro participantes por país presente ao evento, competem individualmente, com até quatro participando para cada país presente no evento. Cada estudante trabalha individualmente, com um computador e sem ajuda externa, como, por exemplo, consulta em material ou a outras pessoas. Para resolver os problemas os competidores têm que criar programas em uma das linguagens permitidas (C, C++ e Pascal). Os programas são testados com diversos casos de teste para avaliar sua eficiência e capacidade de gerar respostas corretas, pontos são dados para cada caso de teste acertado.

3.5 OLIMPÍADA BRASILEIRA DE INFORMÁTICA

A Olimpíada Brasileira de Informática (OBI) é uma competição de programação realizada anualmente desde 1999 pela Sociedade Brasileira de Computação (WIKIPEDIA, 2012).

A OBI está organizada em duas modalidades (SBC, 2012):

1) Modalidade Iniciação – nessa modalidade os alunos concorrem resolvendo problemas de lógica e problemas de computação, sem uso de computador. As questões das provas da modalidade Iniciação são feitas no papel e exigem raciocínio lógico para sua resolução. Essa modalidade se divide em dois níveis:

1.1) Nível 1 - para alunos até o sétimo ano (sexta série) do Ensino Fundamental (ou equivalente)

1.2) Nível 2 - para alunos até o nono ano (oitava série) do Ensino Fundamental (ou equivalente)

2) Modalidade Programação – a prova dessa modalidade exige conhecimento em programação. As provas da modalidade Programação são realizadas com auxílio de computador usando uma linguagem de programação, que pode ser C, C++ ou Pascal. A correção desses problemas é automatizada.

A prova é composta de tarefas de programação com níveis variados de dificuldade: há tarefas mais fáceis, em que um conhecimento mínimo de programação é suficiente, e algumas tarefas mais difíceis, que exigem um conhecimento um pouco mais avançado de programação, com noções de estruturas de dados, em um nível normalmente ensinado em bons colégios técnicos, ou no primeiro ano de cursos superiores de computação ou

engenharia. Essa modalidade está dividida em três níveis:

2.1) Nível Júnior - para alunos até o nono ano (oitava série) do ensino fundamental - essas provas do nível júnior exigem conhecimento muito básico de programação e estrutura de dados simples.

2.2) Nível 1 - para alunos até o segundo ano do ensino médio - essas provas do nível 1 exigem conhecimento básico de programação e de estrutura de dados.

2.3) Nível 2 - para alunos que estão cursando o ensino médio ou que o concluíram há menos de um ano - essas provas do nível 2 exigem conhecimento de técnicas de programação e algoritmos.

Em todas as modalidades os alunos competem individualmente. Cada aluno poderá estar inscrito em apenas uma modalidade.

4 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizado para o desenvolvimento deste trabalho. Os materiais se referem aos recursos utilizados para realizar a modelagem e a implementação. O método são as atividades desenvolvidas para que o objetivo do trabalho fosse alcançado.

4.1 MATERIAIS

Os materiais são as ferramentas, as tecnologias, os ambientes de desenvolvimento e outros que são utilizados para realizar as atividades desde a definição dos requisitos à implantação do sistema. Os materiais utilizados são:

a) Astah Community para a modelagem do sistema;
b) NetBeans versão 6.8 como ambiente de desenvolvimento;
c) PHP versão 5.3.1 como linguagem de programação com JavaScript par auxílio na elaboração da interface;
d) MySQL para o banco de dados;
e) phpMyAdmin para gerenciamento do banco de dados;
f) Apache Tomcat como contêiner *web*.

4.1.1 Astah Community

Astah Community (ASTAH, 2012) é uma ferramenta para modelagem gratuita para projeto de sistemas orientados a objeto. A modelagem é baseada em diagramas com notação UML (*Unified Modeling Language*). A partir da modelagem é possível gerar código em Java, mas apenas para a definição da classe e de seus atributos e métodos. Para que essa geração de código possa ser feita, o diagrama de classes deve estar pronto, as classes bem definidas com a especificação dos seus atributos e os métodos precisam ter os seus parâmetros definidos.

A Figura 2 apresenta um *print screen* da interface principal dessa ferramenta que é basicamente composta por uma área de edição, à direita, na qual são colocados os componentes gráficos para a representação dos modelos. Esses componentes são pré-definidos e ficam na barra imediatamente superior à área de edição. Os componentes

apresentados nessa barra são de acordo com o tipo de modelo de diagrama em edição.

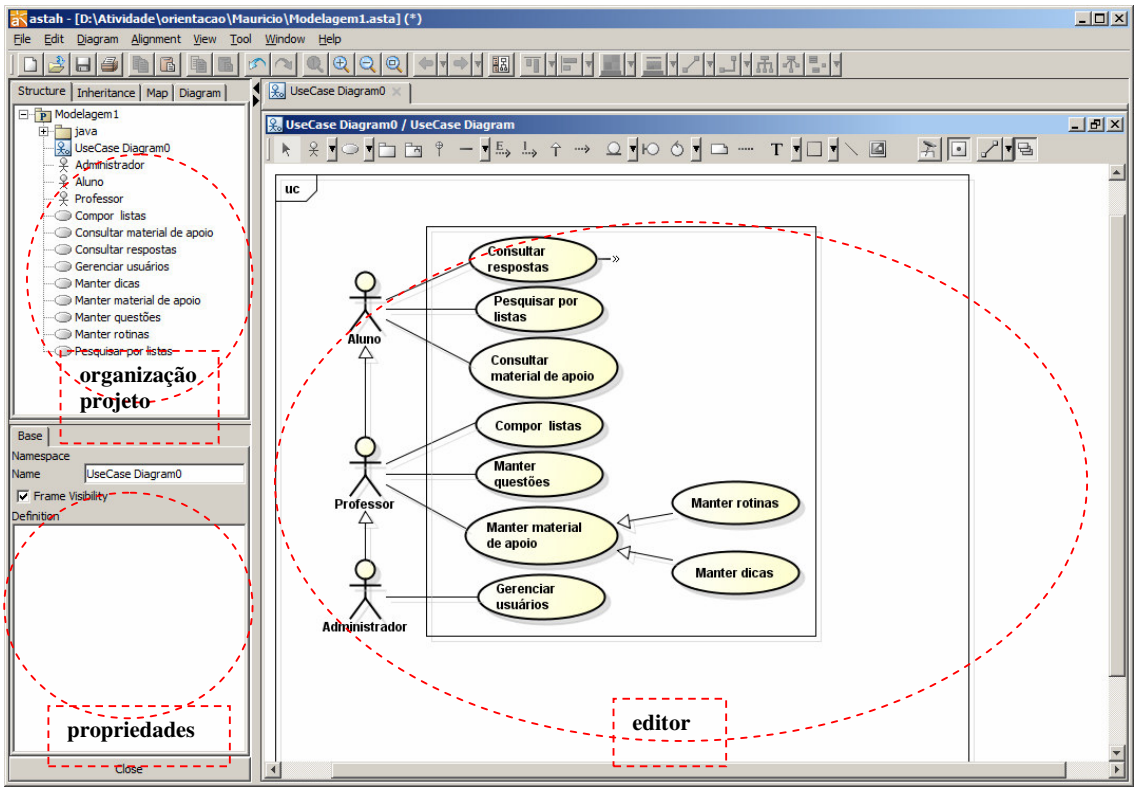

Figura 2 – Ferramenta de modelagem Astah

Na Figura 2 estão marcadas (com círculos pontilhados) as três principais partes da ferramenta Astah Community. Essas partes estão descritas a seguir:

a) Organização do projeto – apresenta as diversas formas possíveis de visualizar o projeto e os diagramas que os compõem, além dos elementos que compõem os diagramas. Essas formas, denominadas visões são:

a.1) *Structure* – que apresenta a organização do projeto como uma espécie de árvore de diretórios. Os diagramas e os elementos que os compõem são apresentados.

a.2) *Inheritance* – exibe as heranças identificadas.

a.3) *Map* – apresenta a visualização completa do diagrama em edição e destacado a parte que está sendo apresentada no momento na área de edição. É possível fazer a seleção da parte do diagrama a ser apresentada na área de edição.

a.4) *Diagram* – apresenta a lista de diagramas do projeto. Além do nome do projeto é apresentado o seu tipo e a sua herança.

b) Visão das propriedades – é a área utilizada para visualizar as propriedades dos elementos do diagrama. As propriedades de um item selecionado são exibidas e podem ser editadas.

c) Editor do diagrama – é a área na qual os diagramas são compostos e também podem ser editados. Ao ser selecionado um determinado diagrama, dos constantes na lista, o mesmo é carregado e todos os seus elementos gráficos são mostrados nesta área.

4.1.2 NetBeans

NetBeans (NETBEANS, 2012) é um ambiente integrado de desenvolvimento (IDE - *Integrated Development Environment*) utilizado para produzir código fonte de linguagens como Java e PHP, por exemplo. É um ambiente de desenvolvimento multiplataforma, uma ferramenta para escrever, compilar, depurar (*debug*) e gerar instaladores. A IDE NetBeans podem estar vinculados bibliotecas, módulos e APIs (*Application Programming Interface*) que são basicamente um conjunto de rotinas, protocolos e ferramentas para a construção de *software*. Facilitando, assim, a implementação de sistemas. Na Figura 3 está um *print screen* da tela principal da IDE NetBeans.

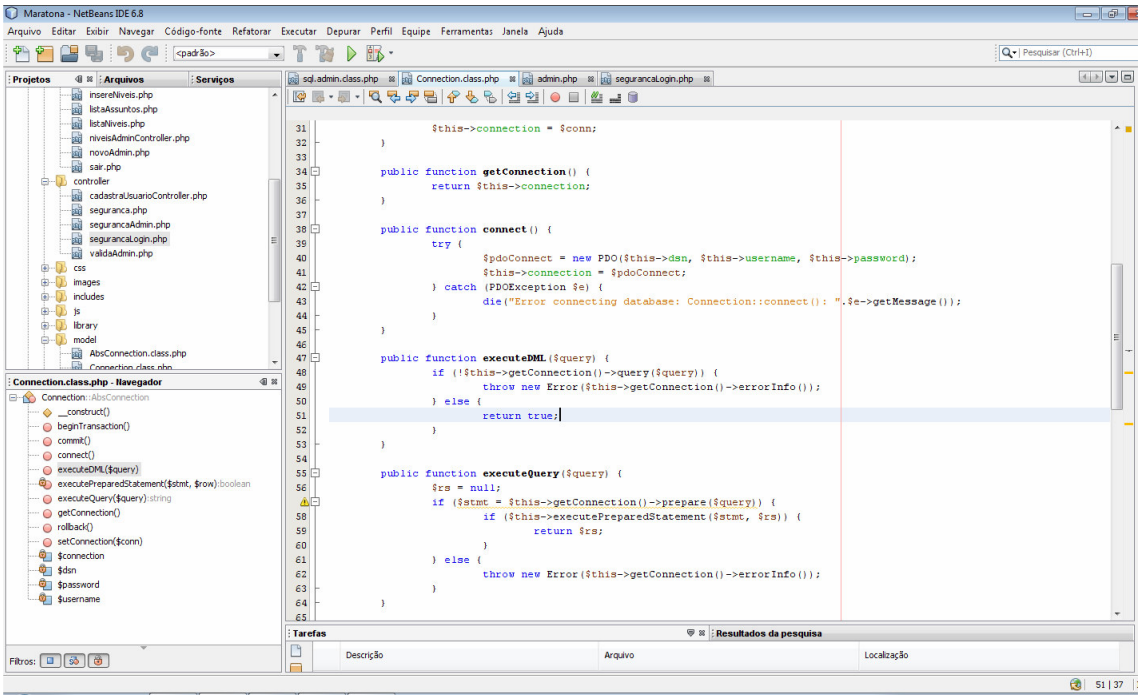

Figura 3 – IDE NetBeans

Na Figura 3, na área mais a esquerda está à organização do projeto. Estão os arquivos desenvolvidos e componentes utilizados no projeto. Na área superior à direita está o editor de código. E na parte inferior direita, estão as abas de saída (resultados) do editor.

4.1.3 Linguagem PHP

PHP é uma linguagem de programação dinâmica para *web* (PHP, 2012). PHP é processada no servidor, retornando para o cliente somente HTML. Assim, o código fonte não é exposto.

A linguagem PHP tem suporte a praticamente todos os bancos de dados existentes no mercado, o que torna simples a integração com aplicações que necessitem desta tecnologia. A linguagem também suporta outros protocolos como SMTP (*Simple Mail Transfer Protocol)*, POP3 (*Post Office Protocol)* e IMAP (*Internet Message Application Protocol*).

4.1.4 phpMyAdmin

phpMyAdmin é uma ferramenta de *software* escrita em PHP para administração do MySQL pela Internet. A partir deste sistema é possível criar e remover bases de dados, criar, remover e alterar tabelas; inserir, remover e editar campos; executar códigos SQL e manipular campos chaves (PHPMYADMIN, 2012).

phpMyAdmin permite realizar uma variedade de operações com MySQL e as mais frequentemente usadas são suportadas pela interface do usuário, incluindo, gerenciamento de bases de dados, tabelas, campos, relacionamentos, índices, usuários e permissões. Além das operações que podem ser realizadas por meio da interface gráfica do phpMyAdmin é possível executar diretamente instruções em SQL.

4.1.5 MySQL

O MySQL é um servidor e gerenciador de banco de dados relacional que utiliza a linguagem SQL (MYSQL, 2012). MySQL possui licença na forma de *software* livre e também paga.

As principais características incorporadas na versão 5 do MySQL (MILANI, 2007) são:

a) Visões - são consultas pré-programadas ao banco de dados que permitem unir duas ou mais tabelas e retornar uma única tabela como resultado. As visões podem ser utilizadas para filtrar informações, exibindo somente os dados específicos de tabelas. As visões também podem ser utilizadas para controle de acesso, permitindo que determinados usuários acessem

dados de determinada visão, mas não as tabelas utilizadas para compor a visão.

b) Cursores - cursores possibilitam a navegação em conjuntos de resultados. Com o uso de cursores é possível navegar pelos registros de uma tabela a partir de laços de repetição.

c) *Information Schema* - são tabelas responsáveis apenas pela organização dos recursos do banco de dados, o dicionário de dados ou os metadados.

d) Transações distribuídas XA - é uma espécie de extensão da ACID (Atomicidade, Consistência, Isolamento, Durabilidade) que fornece a possibilidade de gerenciamento dessas transações realizadas com a união de múltiplos bancos de dados (transações distribuídas) para a execução de uma mesma transação.

e) Integridade referencial - os relacionamentos entre tabelas distintas são gerenciados pelo banco de dados quando de inclusão, alteração e exclusão visando impedir a ocorrência de inconsistências no banco.

f) Clusterização - a clusterização é baseada na integração e sincronismo de dois ou mais servidores com o objetivo de dividir a carga das execuções entre si. Além da sincronização de um *cluster*, é possível especificar um balanceador de cargas. Esse recurso gerenciará o redirecionamento de servidores secundários no caso de parada do funcionamento e balanceará as requisições recebidas pelo *cluster*, distribuindo-as pelos servidores de acordo com a disponibilidade de cada um.

4.1.6 Tomcat

O Tomcat (TOMCAT, 2011) é um contêiner *servlet* utilizado para interpretar aplicações para *web*. Tomcat também pode atuar como servidor *web*, ou funcionar integrado a um servidor *web* dedicado como o Apache ou o IIS (*Internet Information Services*).

Como servidor *web*, Tomcat provê um servidor HTTP puramente em Java. O servidor inclui ferramentas para configuração e gerenciamento, o que também pode ser feito pela edição manual dos arquivos de configuração que são formatados em XML.

4.2 MÉTODO

O processo de modelagem do aplicativo para auxílio no treinamento para maratonas de programação e da sua implementação tem como base o modelo sequencial linear como descrito em Pressman (2008), complementado pelo processo unificado (BLAHA, 2006). A

documentação dos casos de uso foi realizada tendo como base o disposto em Bezerra (2007). Ainda que tenha como base o modelo sequencial linear o desenvolvimento do trabalho não foi realizado atendendo plenamente esse padrão. Inicialmente foram definidos os requisitos principais, obtendo-se uma visão geral do sistema. Esses requisitos foram complementados à medida que o sistema era modelado e implementado.

O processo unificado auxiliou a definir os ciclos interativos de modelagem e implementação. Essa forma de procedimento foi adotada porque os requisitos do sistema não estavam completamente definidos e era necessário estudo de alguns aspectos das tecnologias utilizadas.

O Quadro 1 apresenta os processos (fluxos de trabalho) e as iterações desenvolvidas. As iterações 1 e 2 foram realizadas como trabalho de estágio e as iterações 3 e 4, como trabalho de conclusão de curso.

	1ª iteração	**2ª iteração**	**3ª iteração**	**4ª iteração**
Requisitos	Primeira versão.	Refinamento dos requisitos.		
Análise e projeto	Definição dos casos de uso.	Definição de tabela e classe básicas.	Ajustes na modelagem do sistema.	
Implementação	Interface padrão dos cadastros e do sistema (página *web*).	Implementação de cadastros para exemplificar o emprego do padrão MVC.	Implementação de cadastros restantes.	Implementação da verificação dos arquivos e funcionalidades restantes.
Testes	De modelo de interface realizado pela orientadora do trabalho.	De código realizado pelo autor deste trabalho.	De código realizado pelo autor deste trabalho. De funcionalidades realizadas pela orientadora do trabalho.	De código realizado pelo autor deste trabalho.

Quadro 1 – Iterações definidas

A seguir estão descritas as etapas definidas para o desenvolvimento do aplicativo e as principais atividades de cada uma dessas etapas.

a) Levantamento de requisitos

O levantamento dos requisitos iniciou com a orientadora fornecendo a visão geral do sistema e as funcionalidades pretendidas do ponto de vista do usuário. Esses requisitos definiram a ideia e visão geral do sistema em termos das funcionalidades esperadas para o mesmo.

Esses requisitos permitiram identificar os principais requisitos funcionais e não funcionais. Os requisitos foram complementados à medida que as iterações ocorriam. Várias alterações ocorreram como forma de definir mais adequadamente as funcionalidades do sistema.

c) Análise e projeto do sistema

Com base nos requisitos foram definidos os casos de uso. Esses casos de uso foram documentados gerando informações para a definição do banco de dados e das classes. Em termos de classes e de entidades e relacionamentos do banco de dados inicialmente foram definidas apenas as tabelas implementadas. A modelagem foi complementada à medida que as iterações ocorreram.

d) Implementação

A implementação foi realizada utilizando a IDE Netbeans. Inicialmente foram implementados cadastros simples visando exemplificar o padrão de projetos MVC. Durante a implementação ocorreram testes para verificar se os requisitos estavam sendo atendidos. Em seguida as demais funcionalidades do sistema foram implementadas.

e) Testes

Os testes realizados foram de verificação de código e de funcionalidades e usabilidade. Os testes de verificação de código foram realizados pelo autor deste trabalho e de funcionalidades e usabilidades foram realizados pela orientadora.

5 RESULTADOS E DISCUSSÃO

Este capítulo apresenta o resultado deste trabalho que é um sistema para auxílio no treinamento para maratonas de programação. A implementação visa exemplificar a aplicação do padrão de projetos MVC com a linguagem PHP.

5.1 APRESENTAÇÃO DO SISTEMA

O sistema visa auxiliar no preparo para maratonas de programação. Considera-se como maratona competições que podem denominar-se, inclusive, olimpíadas. Ressalta-se que o sistema não é exclusivamente para maratona. Esse é o seu objetivo principal, mas pode ser utilizado como um repositório de material visando o aprendizado de qualquer assunto e de qualquer área.

O acesso ao sistema será feito por *login*. Assim, é preciso que o usuário esteja cadastrado para poder utilizar o sistema. Há três tipos de usuários, denominados: Aluno, Professor e Administrador. O usuário Aluno pode consultar material, acessar listas de questões e submeter saídas para verificação. O usuário Professor pode compor listas, fazer cadastramento de materiais e exercícios. E o usuário Administrador tem acesso a todas as funcionalidades do sistema, inclusive a manutenção de usuários que é exclusiva desse usuário.

O sistema é composto basicamente por cadastro de questões que compõem listas, de respostas a essas questões e de materiais complementares. As questões são catalogadas por assuntos e esses assuntos permitem definir materiais auxiliares. Visando, assim, fornecer subsídios ao aprendizado do aluno. As listas de questões podem ser de uma maratona específica ou serem compostas pelos professores. Uma mesma questão pode ter respostas distintas em linguagens (de programação ou outra forma de representação de algoritmo) distintas ou em uma mesma linguagem.

5.2 MODELAGEM DO SISTEMA

A seguir a listagem dos requisitos do ponto de vista do usuário. Significa que são as funcionalidades requeridas ao sistema pelo seu solicitante.

a) Acesso ao sistema

O acesso ao sistema ocorrerá por nível de usuário que são:

1) Administrador – todas as funcionalidades do sistema.

2) Professor – todas as funcionalidades, exceto cadastro de professor.

3) Aluno – acesso público para consulta às listas de questões, suas respostas e material de apoio disponibilizado.

Quanto à forma de cadastro de usuários do tipo Professor são indicadas as seguintes como opções possíveis:

1) O usuário faz o seu próprio cadastro e o mesmo é posteriormente validado pelo Administrador;

2) O Administrador faz cadastro, o sistema gera uma senha padrão e no primeiro acesso o usuário altera a senha e complementa o cadastro com os seus dados.

b) Cadastro de questões

1) Cada questão possui uma descrição breve que as identifica. Essa descrição é apresentada para a escolha das questões para compor uma lista de questões.

2) As questões possuem um enunciado. Esse enunciado pode conter uma figura associada e o enunciado pode ser longo (mais de uma página).

3) As questões são categorizadas por nível de dificuldade.

4) As questões possuem assuntos vinculados. Uma mesma questão pode possuir mais de um assunto vinculado. Os assuntos são como palavras-chave que categorizam as questões.

5) As questões possuem material de apoio vinculado. Esses materiais são arquivos de texto, imagens e outros que podem ser utilizados como auxiliares na resolução de uma questão ou para o estudo de um determinado conteúdo.

6) As questões podem possuir rotinas (partes de código: funções, por exemplo) vinculadas. Essas rotinas implementam funcionalidades que podem ser utilizadas na resolução de uma questão. Por exemplo, uma questão que precise de cálculo da raiz quadrada pode ter vinculada uma função que calcula a raiz quadrada.

7) Uma questão pode ter dicas vinculadas. As dicas visam auxiliar na resolução da questão. Uma dica é uma orientação de como determinado aspecto da questão pode ser implementado, orientações e conceitos que estão envolvidos.

8) No cadastro da questão vincular respostas. Não há um cadastro de respostas, as respostas são vinculadas no cadastro da questão. Uma questão pode possui “n” respostas e “n” materiais de apoio vinculados.

c) Cadastro de respostas

1) As questões podem possuir mais de uma resposta, sejam em uma mesma linguagem de programação ou em linguagens distintas. A resposta não precisa estar em uma linguagem de programação, pode estar na forma de algoritmo em alto nível, fluxograma (imagem), texto ou outro.

2) As respostas das questões podem possuir arquivo de entrada para teste e arquivo de saída gerado para verificar a resposta. O sistema verificará se o arquivo de saída gerado pelo usuário é igual ao arquivo de saída vinculado à resposta do sistema.

d) Cadastro de listas

1) Uma lista é um conjunto de questões. Uma lista pode referir-se à prova de uma determinada maratona ou ter sido composta por um usuário Professor. Uma determinada lista pode referir-se a um conjunto de conteúdos que estarão vinculados à lista. Uma lista deve ter uma identificação que a individualize. Essa identificação é utilizada para a visualização de respostas às questões que a compõem.

2) Quando uma lista é acessada o usuário tem a opção de visualizar os conteúdos associados às questões e os respectivos materiais de apoio.

3) As questões podem estar vinculadas às listas, no sentido de compor listas.

e) Cadastro de materiais de apoio

1) Os materiais de apoio são os conteúdos relacionados às questões ou aos assuntos. Os materiais de apoio podem ficar armazenados no banco de dados ou em um sistema de arquivos (pastas) com acesso local ou remoto. Os materiais podem ser disponibilizados sob a forma de link para acesso ao seu conteúdo que pode estar na máquina local, rede local ou Internet, isto é, eles não precisam estar armazenados localmente ou sob controle do sistema desenvolvido.

2) Um material de apoio pode ser no formato texto, figura, código ou mesmo vídeo.

3) Material de apoio inclui dicas. As dicas são orientações específicas e bem pontuais que visam auxiliar na resolução de problemas. As dicas são categorizadas por palavras-chaves e podem estar associadas a questões ou assuntos.

4) Material de apoio também é composto por funcionalidades específicas implementadas. Exemplos de funcionalidade: função para realizar o cálculo da raiz quadrada de um número, função para gerar um vetor com valores aleatórios. Essas funções implementadas podem agilizar a solução de problemas ou ser utilizadas em estudos. Portanto, elas podem estar vinculadas a questões ou assuntos.

5) Dicas é um tipo de material de apoio que tem o objetivo de fornecer

informações/orientações pontuais que possam auxiliar na resolução de exercícios.

6) Rotinas é outro tipo de material de apoio. Rotinas são códigos fonte (funções, classes e outros) que implementam funcionalidades específicas e podem ser reusadas na resolução de exercícios.

f) Consultas/relatórios

1) Relatório de listas com as respectivas questões e opção de mostrar ou não respostas e mostrar ou não material de apoio vinculado à questão. Se optado por mostrar as respostas é apresentado o conteúdo de cada um dos arquivos que contém a resposta da respectiva questão. Se optado por mostrar material de apoio vinculado apresentar os mesmos como *link* para *download*.

2) Consulta de assuntos. O usuário seleciona um determinado assunto e são listados todos os materiais disponíveis para aquele assunto. A lista gerada permitirá o *download* dos materiais.

3) Consulta de questões. Ao consultar uma questão poder ter acesso aos materiais vinculados à mesma. Os materiais vinculados à respectiva questão são apresentados como uma lista e é permitido fazer *download* dos mesmos.

4) Consulta de respostas. Pode ser consultada a resposta a uma questão.

A Figura 4 apresenta o diagrama de casos de uso identificados a partir dos requisitos.

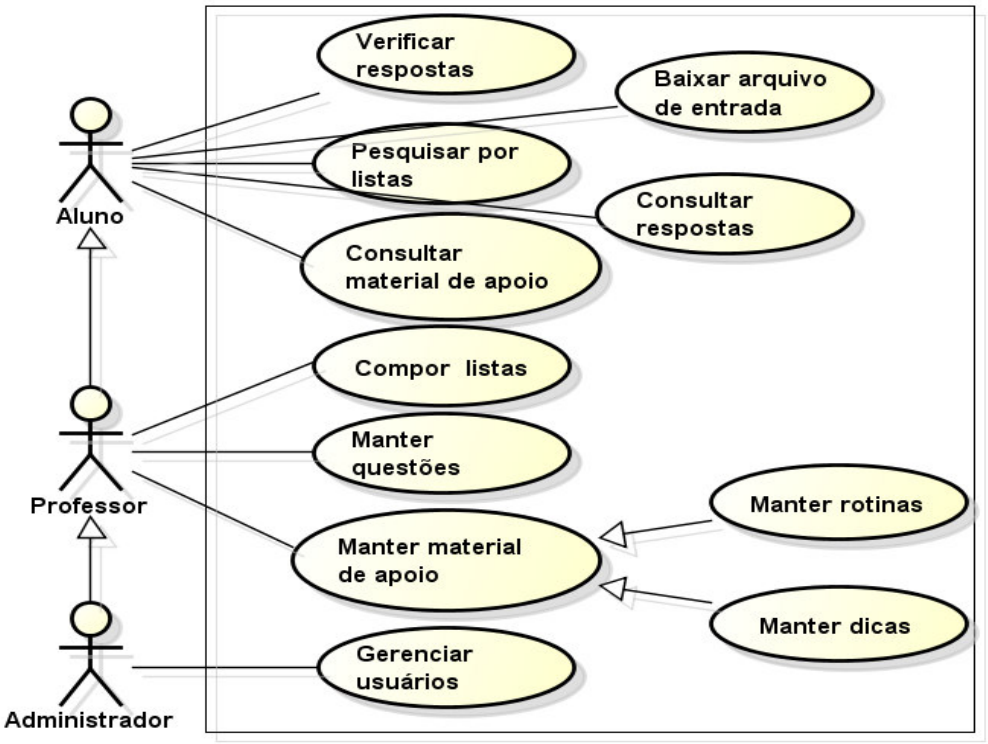

Figura 4 – Diagrama de casos de uso

Os casos de uso apresentados na Figura 4 estão descritos nos Quadros 2 a 11. O Quadro 2 apresenta o caso de uso baixar arquivo de entrada. É o arquivo base para gerar o arquivo de saída pelo algoritmo implementado.

Caso de uso: **Baixar arquivo de entrada** Descrição: O ator Aluno pode baixar (fazer *download*) de arquivo de entrada. Esse arquivo é utilizado para gerar o arquivo de saída pela solução implementada pelo usuário. É um arquivo texto com dados para serem utilizados pelo algoritmo visando gerar o arquivo de saída para verificação se a solução está correta. Ator principal: Aluno Precondições: O ator Aluno está identificado no sistema. Fluxo principal: 1. O Aluno seleciona/indica a questão da qual ele quer baixar o arquivo de entrada. 2. O Sistema disponibiliza o arquivo para *download*. Pós-condições: O arquivo de entrada para verificar a questão foi baixado.

Quadro 2 – Caso de uso baixar arquivo de entrada

O Quadro 3 apresenta o caso de uso verificar respostas. É a verificação da resposta de um exercício. A verificação é feita pela comparação entre o arquivo de saída gerado pelo algoritmo e o arquivo de saída armazenado no sistema. Esse arquivo de saída é gerado a partir do arquivo de entrada descrito no Quadro 2.

Caso de uso: **Verificar respostas** Descrição: O ator Aluno submete um arquivo texto contendo a resposta gerada pelo sistema e o sistema verifica se o arquivo informado pelo usuário é igual (idêntico) ao arquivo de saída vinculado àquela resposta. Ator principal: Aluno Precondições: O ator Aluno está identificado no sistema. Fluxo principal: 1. O Aluno seleciona/indica a questão para a qual ele quer submeter o arquivo de saída para ser avaliado. A seleção é feita pela escolha de uma lista e em seguida pela escolha da questão. Ao ser escolhida a lista, o sistema apresenta todas as questões daquela lista. 2. O Aluno indica a questão para a qual ele quer submeter o arquivo de saída (resposta) para verificação. 3. O Aluno submete o arquivo para *upload* no sistema. 3. O sistema verifica o arquivo e informa se o mesmo é igual ao arquivo armazenado. Pós-condições: O arquivo de resposta da questão foi verificado.

Quadro 3 – Caso de uso verificar respostas

O usuário pode consultar respostas de questões armazenadas no sistema, caso de uso descrito no Quadro 4. Uma mesma questão pode ter mais de uma resposta. Uma resposta pode ser o algoritmo implementado em uma linguagem de programação, em pseudocódigo, uma representação gráfica, um texto explicativo, dentre outros.

Caso de uso: **Consultar respostas**
Descrição: O ator Aluno consulta a resposta de uma determinada questão.
Ator principal: Aluno
Precondições: O ator Aluno está identificado no sistema.
Fluxo principal:

1. O Aluno indica a lista da qual ele quer consultar a resposta a uma determinada questão.
2. O Aluno indica a questão da qual ele quer visualizar a resposta.
3. O sistema apresenta a lista de respostas para a referida questão. No rótulo dessa listagem é apresentada a linguagem/forma de representação. Por exemplo: "Java", "Português estruturado". Uma questão pode ter mais de uma resposta, em uma mesma linguagem de programação ou forma de representação ou em linguagens/formas distintas. Juntamente com a resposta são indicados materiais de apoio.
4. O Aluno seleciona a resposta que quer visualizar.
5. O sistema apresenta ao aluno a resposta que pode apresentada na mesma janela, em uma nova janela, em link para baixar o arquivo ou indicação de endereço para redirecionamento.

Pós-condições: A resposta solicita é apresentada ao aluno.

Quadro 4 – Caso de uso consultar resposta

Uma das principais funcionalidades do sistema está nas listas de exercícios ou problemas disponibilizadas aos usuários. O caso de uso apresentado no Quadro 5 permite pesquisar por listas incluídas no sistema.

Caso de uso: **Pesquisar por listas**
Descrição: O Aluno localiza listas de questões armazenadas.
Ator principal: Aluno
Precondições: O ator Aluno está identificado no sistema.
Fluxo principal:

1. O Aluno escolhe a lista que quer visualizar.
2. Sistema apresenta a lista para o aluno que pode ser visualizada em tela ou baixada (*download*).

Pós-condições:
A lista solicitada é apresentada ao aluno.

Quadro 5 – Caso de uso pesquisar por listas

Os materiais de apoio estão vinculados às questões que compõem as listas. Esses materiais de apoio visam auxiliar o aluno na compreensão de conceitos, por exemplo, para a resolução das questões que compõem as listas. O caso de uso consultar material de apoio é apresentado no Quadro 6.

Caso de uso: **Consultar material de apoio**
Descrição: O aluno consulta materiais de apoio para determinados assuntos.
Ator principal: Aluno
Precondições: O ator Aluno está identificado no sistema.
Fluxo principal:

1. O Aluno indica o assunto do qual quer consultar material de apoio. O assunto,

termo, de consulta é apresentado como uma listagem a partir dos assuntos armazenados no banco de dados.
2. O sistema apresenta a listagem dos materiais cadastrados na base de dados para o assunto indicado.
3. O aluno pode acessar os materiais apresentados na listagem. Esses materiais são apresentados sob a forma de arquivos para *download*, visualização em nova janela, indicação de *link* ou outro.

Pós-condições:
O aluno tem acesso aos materiais relacionados ao termo de consulta indicado.

Quadro 6 – Caso de uso consultar material de apoio

O Quadro 7 apresenta o caso de uso compor lista. Uma lista é composta por um conjunto de questões. O aluno visualiza essas listas para responder as questões propostas.

Caso de uso: **Compor listas**
Descrição: O professor define a composição de uma lista de questões.
Ator principal: Professor.
Precondições: O ator Professor está identificado no sistema.
Fluxo principal:
1. O Professor solicita a inclusão de uma nova lista.
2. O sistema apresenta o formulário para o cadastro de uma nova lista.
3. O Professor inclui as informações solicitadas e seleciona as questões para compor a lista.
4. O sistema informa que a lista composta foi cadastrada.

Pós-condições:
Lista incluída no sistema.

Quadro 7 – Caso de uso compor listas

As questões são cadastradas pelo usuário Professor ou Administrador. O caso de uso manter questões é apresentado no Quadro 8.

Caso de uso: **Manter questões**
Descrição: Questões são cadastradas para posteriormente serem utilizadas na composição de listas.
Ator principal: Professor
Precondições: O ator Professor está identificado no sistema.
Fluxo principal:
1. O Professor solicita a inclusão de uma questão.
2. O sistema apresenta a tela com o formulário para a inclusão de questões.
3. O Professor insere os itens solicitados e solicita a inclusão da questão no banco de dados.
4. O sistema insere a questão no banco de dados e informa o usuário da operação realizada.

Pós-condições:
Questão inserida no banco de dados.

Quadro 8 – Caso de uso manter questões

Além das questões, os materiais de apoio também são cadastrados para poderem ser vinculados às questões. No Quadro 9 está o caso de uso manter material de apoio.

Caso de uso: **Manter material de apoio**
Descrição: Inclusão de material de apoio.
Ator principal: Professor
Precondições: O ator Professor está identificado no sistema.
Fluxo principal:

1. O Professor solicita a inclusão de cadastro de material de apoio.
2. O sistema apresenta a tela com o formulário para o cadastro de material de apoio.
3. O Professor insere as informações solicitadas e solicita a inclusão do cadastro.
4. Sistema insere o cadastro no banco de dados e informa o usuário da operação realizada.

Pós-condições:
Cadastro de material de apoio inserido no banco de dados.

Quadro 9 – Caso de uso manter material de apoio

No Quadro 10 apresentado o caso de uso manter rotinas. Rotinas são funcionalidades implementadas, como, por exemplo, uma função que calcula uma raiz de qualquer índice. Essas rotinas podem ser utilizadas na resolução de outros problemas.

Caso de uso: **Manter rotinas**
Descrição: Inclusão de rotinas. Rotinas são consideradas funcionalidades implementadas sob a forma de funções, classes e outros que podem ser reusadas.
Ator principal: Professor
Precondições: O ator Professor está identificado no sistema.
Fluxo principal:

1. O Professor solicita a inclusão de uma rotina.
2. O sistema apresenta a tela com o formulário para o cadastro de rotinas.
3. O Professor insere as informações solicitadas e solicita a inclusão do cadastro.
4. O sistema insere o cadastro no banco de dados e informa o usuário da operação realizada.

Pós-condições:
Cadastro de rotina inserido no banco de dados.

Quadro 10 – Caso de uso manter rotinas

As dicas visam auxiliar na resolução de problemas e elas estão associadas às perguntas armazenadas no sistema. Dicas são orientações em como entender determinados conceitos, como raciocinar na resolução de problemas, dentre outros. O caso de uso manter dicas é apresentado no Quadro 11.

Caso de uso: **Manter dicas**
Descrição: Inclusão de dicas. Dicas têm o objetivo de auxiliar na resolução de problemas. São orientações que podem facilitar na resolução de problemas.
Ator principal: Professor
Precondições: O ator Professor está identificado no sistema.
Fluxo principal:

1. O Professor solicita a inclusão de cadastro de dicas.
2. O sistema apresenta a tela com o formulário para o cadastro de dicas.
3. O Professor insere as informações solicitadas e solicita a inclusão do cadastro.
4. O sistema insere o cadastro no banco de dados e informa o usuário da operação realizada.

Pós-condições:

Cadastro de dicas inserido no banco de dados.

Quadro 11 – Caso de uso manter dicas

A Figura 5 apresenta o diagrama de entidades e relacionamentos do banco de dados.

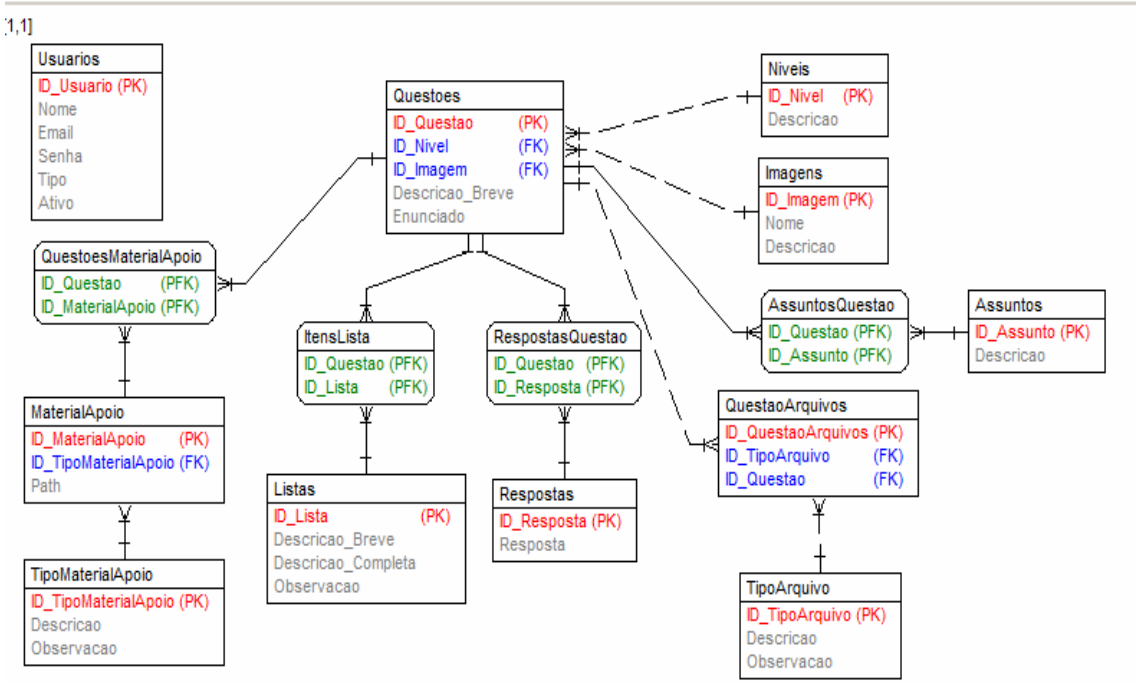

Figura 5 – Diagrama de entidades e relacionamentos do banco de dados

Uma questão pode ter vários arquivos relacionados. Esses arquivos podem conter a resposta da questão em diferentes formatos (um diagrama, código na linguagem Java, código na linguagem C, pseudocódigo, dentre outros). Esses arquivos podem ser casos de teste, conter exemplos de entrada e a respectiva saída. O sistema verificará se é o arquivo de saída vinculado à questão é igual (idêntico) ao arquivo postado pelo usuário para verificação. O arquivo de saída está no formato TXT. Ao ser selecionada a opção para verificação da resposta de uma questão, o usuário informa um arquivo que será verificado com o arquivo TXT armazenado no banco de dados e relacionado à resposta da referida questão.

5.3 DESCRIÇÃO DO SISTEMA

Para acessar o sistema, é necessário fazer o *login*. A Figura 6 apresenta a tela para o acesso do usuário ao sistema.

Figura 6 – Tela de login ao sistema

Para conectar-se ao sistema como Administrador, o usuário deve informar o *email* do Administrador e a senha. Caso os dados informados sejam validados corretamente, é permitido o acesso à tela principal de gerenciamento (Figura 7).

Figura 7 – Tela inicial do sistema

No caso das informações estarem incorretas, haverá um redirecionamento para a tela de *login* novamente. Nessa interface inicial é mostrado o menu principal na parte superior da tela. A partir desse menu são acessadas as opções desejadas. Para cada cadastro será mostrada uma lista com os dados armazenados no banco, com as opções de alterar, excluir e cadastrar novas informações. Há também a opção "Sair" do sistema. Ao clicar nesse *link* é finalizada a sessão atual e o usuário é redirecionado a tela de *login* novamente.

Ao clicar na opção "Níveis" do menu é mostrada a tela de listagem dos níveis

cadastrados no sistema, como mostra a Figura 8. Níveis são utilizados para categorizar os problemas cadastrados no sistema.

Figura 8 – Tela de níveis do sistema

Essa tela permite acesso às opções de incluir, alterar e excluir um registro. Ao clicar em "Novo Cadastro" é mostrada a tela de inclusão. Nessa tela é informada a descrição do nível (Figura 9).

Figura 9 – Tela inclusão de um nível

A tela de alteração de nível é semelhante à tela de inclusão. O campo já vem preenchido com o valor do dado a ser alterado ao ser escolhida a opção "Excluir" na listagem apresentada na Figura 8.

Na Figura 9 apresentada a tela com a listagem de todas as questões cadastradas no sistema. A partir daqui é possível incluir, alterar ou excluir uma questão.

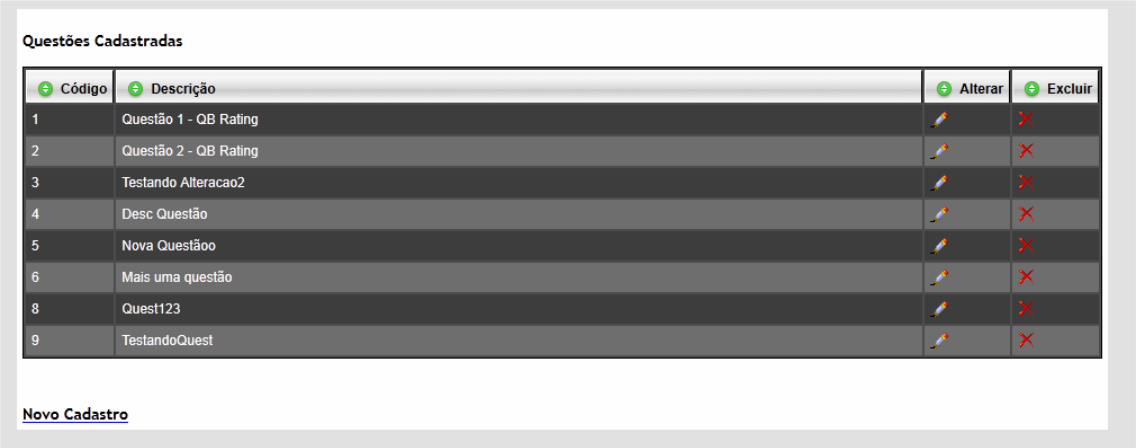

Figura 10 – Tela com a listagem das questões

Ao escolher a opção "Novo Cadastro" é mostrada a tela da Figura 11. Para cadastrar uma nova questão, é necessário indicar uma breve descrição da mesma, o enunciado completo da questão, adicionar imagens à questão (opcional), marcar qual a resposta é correta para essa questão (o arquivo de saída, em formato TXT), além de selecionar um nível e marcar os assuntos relacionados.

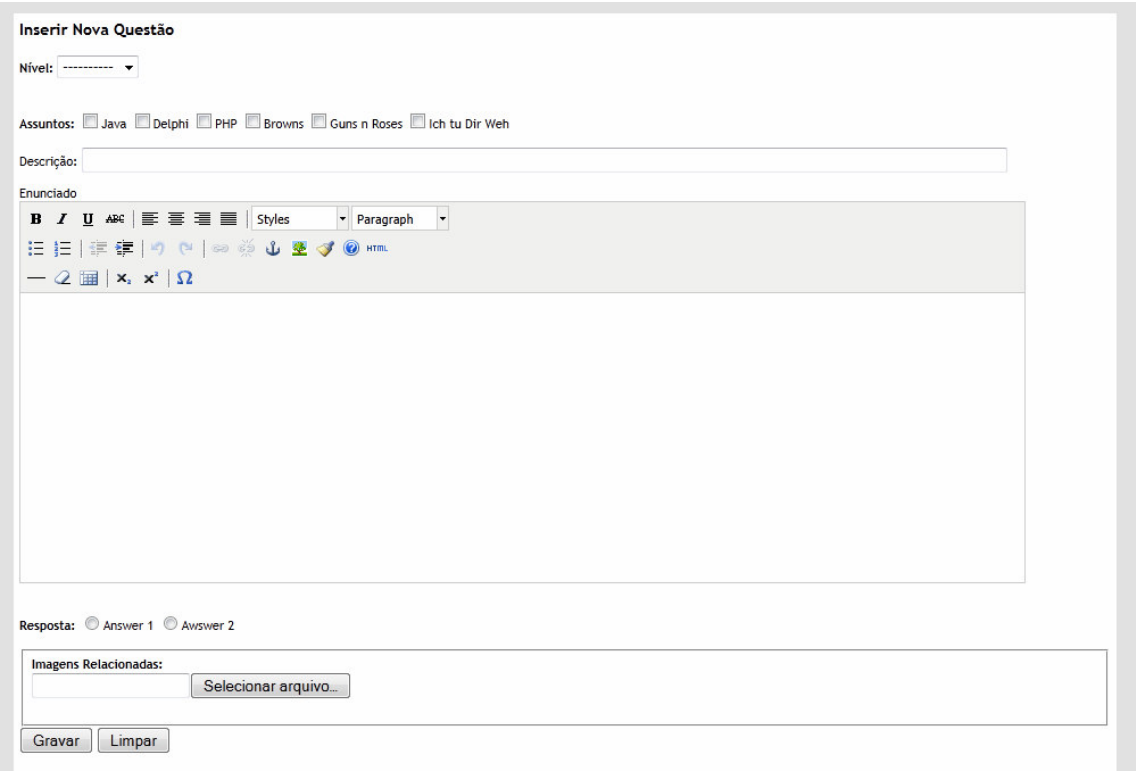

Figura 11 – Tela para cadastro de uma nova questão

A alteração tem basicamente a mesma sequência do cadastro, porém já vem com alguns campos preenchidos como padrão. No caso de haver imagens relacionadas à questão, elas são mostradas no final da página. A tela para alterar questão é apresentada na Figura 12.

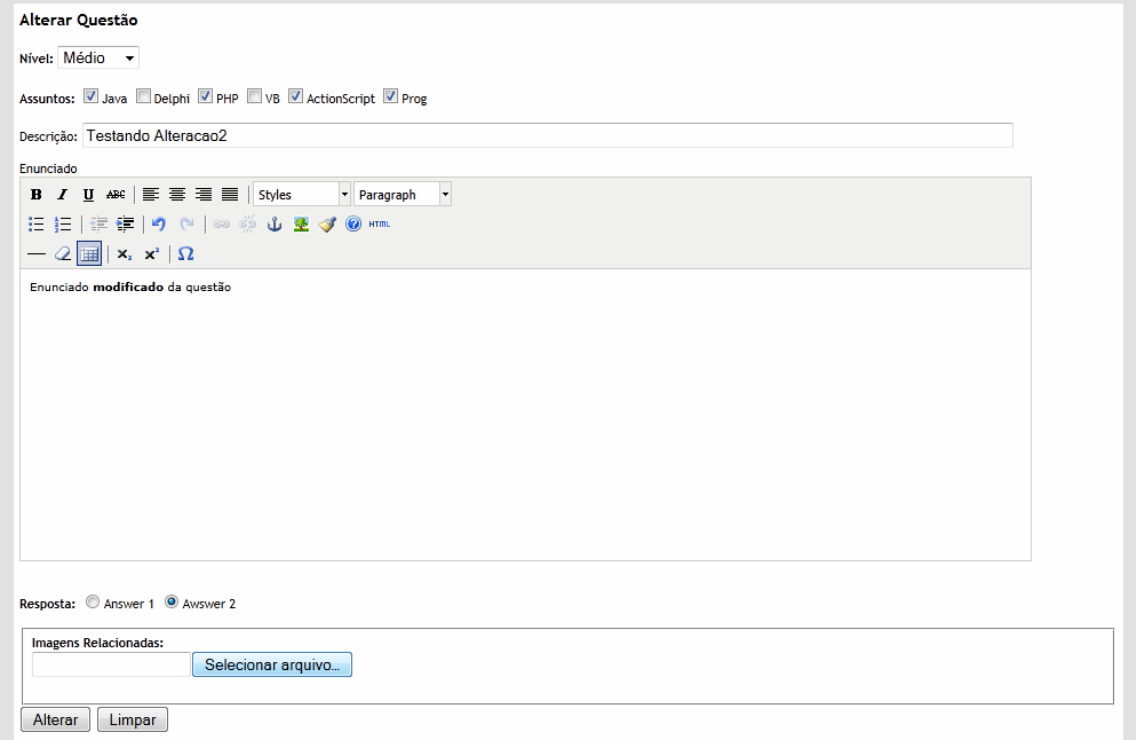

Figura 12 – Tela para alterar uma questão

5.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção é apresentado o processo de autenticação de usuário, criação do menu do sistema e a estrutura MVC utilizada no desenvolvimento do projeto. O MVC tem por objetivo organizar a divisão entre as camadas da aplicação, além da interação entre elas. A Figura 13 apresenta a organização do projeto utilizando o MVC.

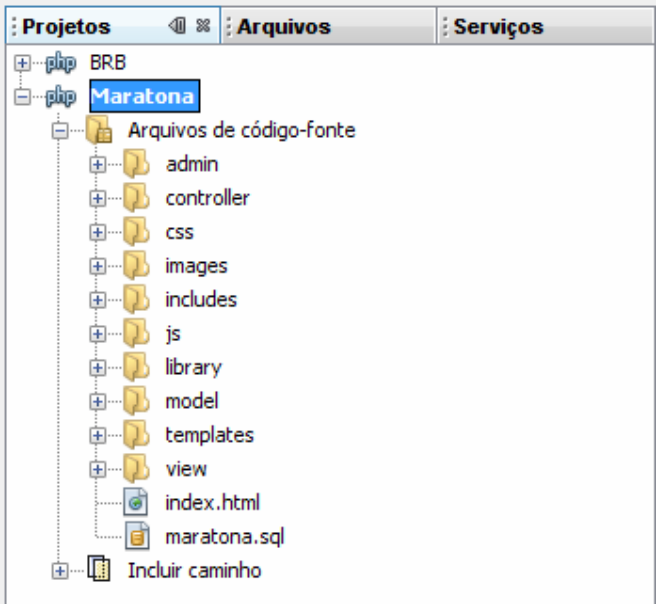

Figura 13 – Tela inclusão de um nível

O processo de autenticação do usuário ocorre por meio da utilização do *email* e senha cadastrados no banco de dados. Esse acesso é feito por meio do arquivo de controle **validaAdmin.php**.

Primeiramente são recuperados os dados digitados pelo usuário, em seguida a senha é criptografada e é executada a função **validaAdmin()**. Nessa função é executada a *query* para verificar os dados informados. Caso a validação seja feita, são gravados os dados na sessão e o usuário é redirecionado para a interface principal da aplicação. Se os dados passados não forem validados o usuário é redirecionado novamente para a tela de *login* do sistema. O código para validação de acesso é apresentado na Listagem 1.

```
<?php ob_start();

require 'seguranca.php';

// Inclui o arquivo com o sistema de segurança
include("segurancaAdmin.php");

// Verifica se um formulário foi enviado
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
// Salva duas variáveis com o que foi digitado no formulário
// Detalhe: faz uma verificação com isset() pra saber se o campo foi
preenchido
    $usuario = (isset($_POST['email'])) ? anti_injection($_POST['email']) :
'';
    $senha = (isset($_POST['senha'])) ? anti_injection($_POST['senha']) :
'';

    $senha = md5($senha);
    $senha = sha1($senha . sha1($senha));
    $senha = md5($senha . md5($senha));
```

```
    $senha = sha1($senha . sha1($senha));

// Utiliza uma função criada no seguranca.php pra validar os dados
digitados
    if (validaAdmin($usuario, $senha, $admin) == true) {
// O usuário e a senha digitados foram validados, manda pra página interna
        header("Location: ../admin/index.php");
    } else {
// O usuário e/ou a senha são inválidos, manda de volta pro form de login
//  Para alterar o endereço da página de login, verifique o arquivo
seguranca.php
        expulsaVisitante();
    }
}
?>
```

Listagem 1 – Validação de acesso do administrador

Em todas as telas do sistema, é exibido o menu principal. Por isso esse menu foi feito como um *template* padrão. Nesse *template* ainda há mais uma função de segurança para fazer controle do acesso ao sistema. Essa função é chamada **protejePagina()**, apresentada na Listagem 3, é executada sempre ao se tentar acessar outra página. Ela simplesmente verifica se os dados da sessão estão corretos, se houver algum problema, é executada a função **expulsaVisitante()**. Essa função remove as variáveis da sessão e redireciona para a página de acesso ao sistema. O *template* do menu é apresentado na Listagem 2.

```
<?php include_once ('../controller/segurancaAdmin.php'); ?>
<?php include_once ('../includes/config.php'); protegePagina();?>
<div id="main">
  <div id="cabec">

  </div>

  <div id="menu">
    <ul id="menu">
    <li>
    <a href="index.php" title="Entrada no site"> Inicial </a>
    </li>
    <li>
    <a href="?AdminController.php?all=1" title="?"> Questões </a>
    </li>
    <li>
    <a href="?AdminController.php?all=1" title="?"> Respostas  </a>
    </li>
    <li>
    <a href="niveisAdminController.php?all=1" title="Niveis">Níveis  </a>
    </li>
    <li>
    <a  href="assuntosAdminController.php?all=1"  title="Assuntos">Assuntos
</a>
    </li>
    <li>
    <a href="?AdminController.php?all=1" title="?">Listas</a>
    </li>
    <li>
    <a href="?AdminController.php?all=1" title="?">Apoio </a>
```

```
    </li>
    <li>
    <a href="?AdminController.php" title="?">Gerenciar Usuários </a>
    </li>
    <li>
    <a href="formUpload.php" title="Upload">Imagens</a>
    </li>
    <li>
    <a href="novoAdmin.php" title="?">Novo Admin</a>
    </li>
    </ul>
  </div>

    <div id="utility_nav">

        <span id="account_links" >
            <?php        if        (!isset($_SESSION['usuarioID'])        OR
!isset($_SESSION['usuarioNome'])) {   ?>
                <a
href="../admin/admin.php">Login<?=$_SESSION['usuarioNome']?></a>
            <?php }  else{ ?>
                <a                      href="../admin/sair.php">Bem-Vindo,
<?=$_SESSION['usuarioNome'] ?> (sair)</a>
            <?php }  ?>
        </span>

         
    </div>
```

Listagem 2 – Template do menu

A Listagem 3 apresenta a função **protegePagina**().

```
function protegePagina() {
    global $_SG;

    if  (!isset($_SESSION['usuarioID'])  OR  !isset($_SESSION['usuarioNome'])
OR ($_SESSION['ehAdmin'] != 2)) {
// Não há usuário logado, manda pra página de login
        expulsaVisitante();
    }        else        if        (!isset($_SESSION['usuarioID'])        OR
!isset($_SESSION['usuarioNome'])) {
// Há usuário logado, verifica se precisa validar o login novamente
        if ($_SG['validaSempre'] == true) {
// Verifica se os dados salvos na sessão batem com os dados do banco de
dados
            if                    (!validaUsuario($_SESSION['usuarioLogin'],
$_SESSION['usuarioSenha'])) {
// Os dados não batem, manda pra tela de login
                expulsaVisitante();
            }
        }
    }
}
```

Listagem 3 – Função para proteger a página

Na Listagem 4 é apresentado o código da função **expulsaVisitante()**. Essa função é definida com o objetivo de redirecionar o usuário para a página de *login* novamente em caso de o usuário inserir informação de *login* e/ou senha incorretos.

```
function expulsaVisitante() {
    global $_SG;

// Remove as variáveis da sessão (caso elas existam)
    unset($_SESSION['usuarioID'],                $_SESSION['usuarioNome'],
$_SESSION['usuarioLogin'], $_SESSION['usuarioSenha'], $_SESSION['ehAdmin'],
$_SESSION['AdminOn']);

// Manda pra tela de login
    header("Location: " . $_SG['paginaLogin']);
}
?>
```

Listagem 4 – Função para redirecionar página em login incorreto

A página principal do sistema se encontra na camada *View*. Ela possui o menu superior que contém as chamadas para a camada *Controller*. Por exemplo, ao clicar na opção "Níveis" do menu, é chamado o arquivo de controle **niveisAdminController.php**. Este arquivo está preparado para executar três ações diferentes, são elas: selecionar todos os níveis cadastrados, excluir um determinado registro e também fazer a inclusão de um novo nível.

A ação a ser executada depende da variável que e passada para esse controle. Ao clicar no menu principal esse controle é feito utilizando a variável "*all*", que indica que será feita a captura de todos os níveis cadastrados.

Na tela de listagem dos níveis é possível selecionar a opção de excluir ou alterar determinado registro e também incluir um novo nível. Caso o usuário escolha a opção de incluir um novo registro, será exibida a tela de cadastro de nível. Após inserir a descrição do novo nível e clicar no botão gravar, será executada a ação de inclusão. Se o método de passagem de parâmetros for **$_POST** o controle apanhará o conteúdo passado na variável "descNivel" e fará a inclusão após instanciar um objeto da classe **sql.admin.class**. Essa classe fica na camada modelo da aplicação.

Caso o usuário escolha excluir um nível cadastrado, por exemplo, ao clicar na opção excluir, serão enviadas para o controle as variáveis "remove" com o valor de 1 e também a "idNivel", que contém o código do respectivo nível. Após recuperar o conteúdo das variáveis o controle executará a ação de excluir, também da classe **sql.admin.class**.

A Listagem 5 apresenta o código de níveis **AdminController.php()**.

```
<?php ob_start();
include_once  ('../controller/segurancaAdmin.php');
include "../model/sql.admin.class.php";
include '../includes/config.php';
require("../library/WideImage/WideImage.inc.php");

$objNiveis = new Admin();
$objInsere = new Admin();

if ($_POST) {

    $descNivel = $_POST['descNivel'];

    $objInsere->insereNivel($descNivel);

    $qryNiveis = $objNiveis->selecionaNiveis();

    ?>

    <script language="JavaScript" type="text/javascript">
        alert ("Nível adicionado com sucesso!!")
    </script>

     <?php
    include "listaNiveis.php";

    } if (($_GET['remove'] == '1') && ($_GET['idNivel'] <> '')){

        $idNivel = $_GET['idNivel'];
        $idUsuario = $_SESSION['usuarioID'];

        if(($idUsuario != '') && ($idNivel != '')){

            try {
                $qryNiveis = $objNiveis->deleteNivel($idNivel);
                ?>

                <script language="JavaScript" type="text/javascript">
                    alert ("Exclusão efetuada com sucesso!!")
                </script>

                <?php
            } catch (Exception $e) {
                print_r($e);
            }

        include "index.php";
      }
  } if($_GET['all'] == '1'){
        $idUsuario = $_SESSION['usuarioID'];
            try {
                $qryNiveis = $objNiveis->selecionaNiveis();
                ?>
                <?php
            } catch (Exception $e) {
                print_r($e);
            }
        include "listaNiveis.php";
      }
?>
```

Listagem 5 – Função AdminController.php()

A Listagem 6 apresenta o código da função **sql.admin.class().**

```
<?php
class Admin {
    public function selecionaNiveis($id = null) {

        $sql = "SELECT * FROM ‘niveis’";

        if(($id != null)&&($id != '')) {
        $sql .= " WHERE ‘ID_Nivel’ = '$id'"; }

        $result = mysql_query($sql) or die(@mysql_error());

        return $result;
    }
    public function insereNivel($descNivel) {

        $sql  = " INSERT INTO 'niveis' ( ";
        $sql .= " 'Descrição') ";
        $sql .= " VALUES ('$descNivel')";

        mysql_query($sql) or die(@mysql_error());

    }
    public function alteraNiveis($idNivel, $descNivel) {

        $sql  = " Update 'niveis' SET ";
        $sql .= " 'Descricao' = '$descNivel'";
        $sql .= " WHERE ('ID_Nivel' = '$idNivel')";

        mysql_query($sql) or die(@mysql_error());

    }
    public function deleteNivel($idNivel) {

        $sql = "DELETE FROM 'niveis' WHERE 'ID_Nivel' = '$idNivel'";

        $result = mysql_query($sql) or die(@mysql_error());

        return $result;
    }
}
```

Listagem 6 – Função sql.admin.class()

Já na opção de alterar é utilizado um controle diferente, chamado **alteraNiveisController**. Quando o usuário selecionar a opção alterar, o controle receberá o código e a descrição do nível a ser alterado. O usuário então após editar os dados, clica em gravar e o controle recebe os dados pelo método **$_POST**, atualiza os valores no banco de dados e em seguida seleciona todos os níveis cadastrados e redireciona o usuário para a tela de listagem. O código **alteraNiveisController.php** é apresentado na Listagem 7.

```
<?php
include_once  ("../controller/segurancaAdmin.php");
include_once ("../model/sql.admin.class.php");
include ("../includes/config.php");
```

```
require("../library/WideImage/WideImage.inc.php");

protegePagina();
$objNiveis = new Admin();
$objAltera = new Admin();

if ($_SERVER['REQUEST_METHOD'] == 'GET') {

    $idNivel = $_GET['idNivel'];

        if(($_GET['altera'] == '1') && ($_GET['idNivel'] != '')) {

            $qryNiveis = $objNiveis->selecionaNiveis($idNivel);

            include ("alteraNiveis.php");

            }
            else {
                include "listaNiveis.php";
            }

} if ($_POST) {

    $idNivel = $_POST['idNivel'];
    $descNivel = $_POST['descNivel'];

    $objAltera->alteraNiveis($idNivel, $descNivel);

    $qryNiveis = $objNiveis->selecionaNiveis();
    ?>

    <script language="JavaScript" type="text/javascript">
        alert ("Nível alterado com sucesso!!")
    </script>

    <?php

    include "listaNiveis.php";
}
?>
```

Listagem 7 – Função alteraNiveisController.php()

Na listagem 8 é apresentado o código do arquivo de controle das questões. Esse código está comentado facilitando o seu entendimento.

```
//define os arquivos que serão utilizados
<?php ob_start();
include_once  ('../controller/segurancaAdmin.php');
include "../model/sql.admin.class.php";
include '../includes/config.php';
require("../library/WideImage/WideImage.inc.php");

//====================================

//instancia os objetos da classe admin (- da camada de modelo)
$objQuestoes = new Admin();
$objImgQuestao = new Admin();
$objUltImg = new Admin();
$objInsere = new Admin();
```

```
$objInsereAss = new Admin();
$objInsereResp = new Admin();
$objInsereImg = new Admin();


//==========================================

if ($_POST) { //verifica se o método foi o POST, no caso seria a inclusão
de um novo registro (questão).

    //recupera as variáveis recebidas
    $idNivel = $_POST['idNivel'];
    $descQuest = $_POST['descQuest'];
    $enunciado = $_POST['enunciado'];
    $imagem = $name;

   //começa a trabalhar com a camada de modelo, fazendo a inclusão da
questão no banco de dados.
    $objInsere->insereQuestao($idNivel,$descQuest,$enunciado);

    $qryImgQuestao = $objImgQuestao->selecionaUltimaQuestao();

    while ($row = mysql_fetch_array($qryImgQuestao)) {

       $idQuest = $row['ID_Questao']; //pega o id do ultimo cadastro
    }

 //para cada assunto selecionado na camada view é executada uma inclusão no
banco
    foreach ($_POST['idAssunto'] as $idAssunto) {
        $objInsereAss->insereAssuntoQuestao($idAssunto,          $idQuest);
//inserir os assuntos relacionados
    }
//inserir as respostas...
    if(isset($_FILES['ufileR']['name'])){

      $quant   =   count($_FILES['ufileR']['name']);   //recupera   qtde   de
registros

      for($i = 0; $i < $quant; $i++) {    //percorre todas as respostas
        $tmpName = $_FILES['ufileR']['tmp_name'][$i];
        $arqNome            =            md5(uniqid(time()))            .
strrchr($_FILES['ufileR']['name'][$i], ".");
        $newName = "../xFiles/respostas/" . $arqNome;

        if(!is_uploaded_file($tmpName) ||
                          !move_uploaded_file($tmpName, $newName)){
        } else {
           $objInsere->insereResposta($resposta, $arqNome);
           $qryUltResp  =  $objUltResp->selecionaUltimaResp();  //seleciona
ultimo inserido
           while ($row = mysql_fetch_array($qryUltResp)) {
             $idResp = $row['ID_Resposta']; //pega o id do ultimo
           }
           $objInsereResp->insereRespostaQuestao($idResp, $idQuest);

        } //fim else

      } //fim for
    } //fim insercao respostas
```

```
//aqui seria o cadastro das imagens, começa a verificar se foi selecionada
alguma imagem pelo usuario.

    if(isset($_FILES['ufile']['name'])){
       //echo "Uploading: ".$_FILES['ufile']['name']."<br />";
      $quant   =   count($_FILES['ufile']['name']);   //recupera   qtde   de
registros

      for($i  =  0;  $i  <  $quant;  $i++)  {      //percorre  todas  as  fotos
recebidas
        $tmpName = $_FILES['ufile']['tmp_name'][$i];
        $imgNome            =            md5(uniqid(time()))            .
strrchr($_FILES['ufile']['name'][$i], ".");
        $newName  =  "../images/questoes/"  .  $imgNome;  //diretório  onde
ficará gravada a imagem, mais o nome gerado com base na hora, para evitar
nomes repetidos.

// faz o upload para o servidor...
        if(!is_uploaded_file($tmpName) ||
                            !move_uploaded_file($tmpName, $newName)){
            echo "Falha no upload " . $_FILES['ufile']['name'][$i] .
                 "<br>Nome Temporário: $tmpName <br>";
        } else {
           $objInsereImg->insereImagens($imgNome);  //caso  o  upload  seja
feito corretamente é gravado no banco de dados a imagem relacionada a
questão
           $qryUltImg   =   $objUltImg->selecionaUltimaImg();   //seleciona
ultimo inserido
           while ($row = mysql_fetch_array($qryUltImg)) {
             $idImg = $row['ID_Imagem']; //pega o id do ultimo
           }
           $objInsereImg->insereImagensQuestao($idQuest, $idImg);
           $image = wiImage::load($newName);
           list($largura,  $altura)  =  getimagesize($newName);  //pequeno
tratamento para evitar imagens com resolução muito grande.
           if ($largura > 800) {
             $image = $image->resize(800, 600);
           }

           $image->saveToFile($newName);

        }

      } //fim for
    } //fim if ISSET //else {

    $qryQuestoes   =   $objQuestoes->selecionaQuestoes();   //seleciona   as
questões para redirecionar a página principal

    ?>

    <script language="JavaScript" type="text/javascript">
        alert ("Questão adicionada com sucesso!!")
    </script>

     <?php
    include "listaQuestoes.php";  // faz o redirecionamento

   //verifica se foi passada a variável de exclusão
    } if (($_GET['remove'] == '1') && ($_GET['idQuestao'] <> '')){
```

```
        $idQuestao = $_GET['idQuestao'];
        $idUsuario = $_SESSION['usuarioID'];

        if(($idUsuario != '') && ($idQuestao != '')){

   // caso as variáveis não estejam vazias, a exclusão é efetuada. A
questão é excluída das listas, das respostas e a questão em si em seguida.
            try {
                $qryQuestao = $objQuestoes->deleteQuestaoLista($idQuestao);
                $qryQuestao                =                $objQuestoes-
>deleteQuestaoResposta($idQuestao);
                $qryQuestao = $objQuestoes->deleteQuestao($idQuestao);
                ?>

                <script language="JavaScript" type="text/javascript">
                    alert ("Exclusão efetuada com sucesso!!")
                </script>

                <?php
            } catch (Exception $e) {
                print_r($e);
            }

        include "index.php";
      }

  } if($_GET['all'] == '1'){ // verifica se foi passada a variável para
fazer a seleção de todas as questões

        $idUsuario = $_SESSION['usuarioID'];

            try {
                $qryQuestoes = $objQuestoes->selecionaQuestoes();
                ?>

                <?php
            } catch (Exception $e) {
                print_r($e);
            }

        include  "listaQuestoes.php";  //redireciona  para  a  listagem  das
questões

      }


?>
```

Listagem 8 – Controle das questões

A Listagem 9 apresenta algumas funções implementadas na camada de modelo. São funções para inserir, alterar e excluir questão e excluir resposta.

```
    public function insereQuestao($idNivel, $descQuest, $enunciado) {

        $sql  = " INSERT INTO 'questoes' ( ";
        $sql .= " 'ID_Nivel', 'Descricao_Breve', 'Enunciado') ";
        $sql .= " VALUES ('$idNivel', '$descQuest', '$enunciado')";

        mysql_query($sql) or die(@mysql_error());
```

```
    }

    public function alteraQuestao($idQuestao, $idNivel, $descricao,
$enunciado) {

        $sql  = " Update 'questoes' SET ";
        $sql   .=   "   'ID_Nivel'   =   '$idNivel',   'Descricao_Breve'   =
'$descricao', 'Enunciado' = '$enunciado'";
        $sql .= " WHERE ('ID_Questao' = '$idQuestao')";

        mysql_query($sql) or die(@mysql_error());

    }

    public function deleteQuestao($idQuestao) {

        $sql = "DELETE FROM 'questoes' WHERE 'ID_Questao' = '$idQuestao'";

        $result = mysql_query($sql) or die(@mysql_error());

        return $result;
    }

    public function deleteQuestaoLista($idQuestao) {

        $sql   =   "DELETE   FROM   'itenslista'   WHERE   'ID_Questao'   =
'$idQuestao'";

        $result = mysql_query($sql) or die(@mysql_error());

        return $result;
    }

    public function deleteQuestaoResposta($idQuestao) {

        $sql  =  "DELETE  FROM  'respostasquestao'  WHERE  'ID_Questao'  =
'$idQuestao'";

        $result = mysql_query($sql) or die(@mysql_error());

        return $result;
    }
```

Listagem 9 – Funções para operações no banco de dados

O código para a correção da questão é apresentado na Listagem 10. A correção se refere a verificar se o arquivo no formato TXT informado pelo usuário é idêntico ao arquivo de resposta (com a mesma extensão), armazenado no banco de dados como a resposta correta. A verificação é simples e é baseada na comparação dos arquivos para identificar se os mesmos são idênticos.

```
//Inclusão dos arquivos utilizados
<?php ob_start();
include_once  ('../controller/seguranca.php');
include "../model/sql.class.php";
include '../includes/config.php';
require("../library/WideImage/WideImage.inc.php");
```

```
    require_once (ARQ_SEGURANCA);
    require_once (DIR_MODEL . 'Connection.class.php');
    require_once (ARQ_SQL);


if ($_POST) { //verifica o método

//recupera os valores das variáveis da camada view
    $idL = $_POST['idL'];
    $idQ = $_POST['idQ'];

//verifica se foi selecionado um arquivo com a resposta
    if(isset($_FILES['ufile']['name'])){
      $quant = count($_FILES['ufile']['name']); //recupera qtde de
registros

      for($i = 0; $i < $quant; $i++) {
        $tmpName = $_FILES['ufile']['tmp_name'][$i];
        $arqNome = md5(uniqid(time())) .
strrchr($_FILES['ufile']['name'][$i], ".");
        $newName = "../xFiles/respostasTemp/" . $arqNome;

//faz o upload da resposta para uma pasta temporaria do servidor
        if(!is_uploaded_file($tmpName) ||
                            !move_uploaded_file($tmpName, $newName)){
        } else {

//começa a conexão com o banco
           $conn = new Connection();

            try {
                $conn->connect();
            } catch (Exception $e) {
                print_r($e);
            }

            try {
                $conn->beginTransaction();
            } catch (Exception $e) {
                print_r($e);
            }

//seleciona a resposta da questão no banco de dados e abre o arquivo
de resposta salvo no servidor na hora do cadastro da resposta pelo
administrador. Em seguida pega o arquivo enviado pelo aluno.
             try {
                $resResposta = $conn-
>executeQuery(selecionaResposta($idQ));
                $arqResposta = $resResposta[0]['ArquivoSaida'];
                $arquivoR = file( "../xFiles/respostas/".$arqResposta
, FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES );
                $arquivoA = file( "../xFiles/respostasTemp/".$arqNome
, FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES );


// faz a comparação dos arquivos
                if ($arquivoR != $arquivoA) { ?>

                    <script language="JavaScript"
type="text/javascript">
                    alert ("Resposta Incorreta!!")
```

```
                    </script>

                <?php
                 } else { ?>

                    <script language="JavaScript"
type="text/javascript">
                    alert ("Resposta Correta!!")
                    </script>

                 <?php
                 }

                 unlink($newName); // exclui a resposta enviada pelo
aluno da pasta temporária do servidor.
              } catch (Exception $e) {
                    print_r($e);
              }

             try {
                 $conn->rollback();
             } catch(Exception $e) {
                    print_r($e);
             }
         } //fim else

       } //fim for
     } //fim if ISSET
 }
?>
```

Listagem 10 – Comparação de arquivos de resposta da questão

6 CONCLUSÃO

O objetivo definido para este trabalho foi o de implementar um sistema que visa auxiliar no preparo de alunos para realização de maratonas de programação. Esse objetivo foi alcançado porque um sistema *web* foi implementado visando disponibilizar listas de exercícios e materiais, bem como verificar a saída gerada pelo programa implementado. A verificação da saída é baseada na comparação de um arquivo texto armazenado (cadastrado) no sistema com um arquivo texto submetido pelo usuário.

Considerando que o sistema implementado é para *web* e visa o treinamento para maratonas de programação, o referencial teórico apresentado está centrado aplicações *web* e no padrão MVC. A especificação conceitual desse padrão foi utilizada para separar a implementação da interface com o usuário, da lógica de negócio e das operações realizadas com o banco de dados. Outro capítulo de referencial teórico foi definido para apresentar sobre maratonas de programação.

Em seguida foram apresentadas as tecnologias e as ferramentas utilizadas na modelagem e na implementação do sistema. Nesse capítulo também foi apresentado o método, como a sequência de atividades utilizadas para realizar a definição da modelagem e da implementação. Essas atividades foram organizadas em iterações visando alcançar de forma mais adequadas os objetivos definidos para o sistema.

Com o sistema implementado verificou-se que o desenvolvimento de um sistema para auxílio no preparo para participação em maratonas de programação (inclui olimpíadas) é importante porque os estudantes têm um repositório de problemas, soluções e orientações para resolvê-los. Assim, os estudantes podem ser autodidatas na busca do aprendizado de algoritmos.

A utilização do modelo MVC ajudou principalmente na organização do projeto. Dessa maneira fica mais fácil fazer as alterações necessárias, já que graças ao uso de camadas, é possível modificar, por exemplo, os dados da lógica de negócio sem que ocorram alterações na lógica de apresentação ou interação com o usuário. A própria IDE NetBeans utilizada também facilita, pois é possível visualizar a estrutura do projeto, além das dicas que são sugeridas durante a programação.

A maior dificuldade encontrada foi na elaboração do banco de dados, já que no estágio inicial do projeto ainda não estavam definidos todos os requisitos pretendidos para o sistema e também não estava claro como alguns processos ocorreriam, a exemplo da verificação dos arquivos de saída.

REFERÊNCIAS

ASTAH. **Astah community**. Disponível em: <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: 18 fev. 2012.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 2ª ed. Rio de Janeiro: Elsevier, 2007.

BLAHA, Michael, JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2.** 2.ed. Rio de Janeiro: Elsevier, 2006

BUSCHMANN, Frank, MEUNIER; Regine; ROHNERT, Hans; SOMMERLAD, Peter; STAL, Michael. **Pattern-oriented software architecture**, vol. 1: A System of Patterns. John Wiley & Sons, 1996.

GAMMA, Erich; JOHNSON, Ralph, 1948-; HELM, Richard; VLISSIDES, John. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2004.

GARLAN, David; SHAW, Mary. **An introduction to software architecture**. SEI Relatório Técnico CMU/SEI-94-TR-21, 1994.

GERMOGLIO, Guilherme. **Arquitetura de software**, Texas: Rice University, 2010.

GONÇALVES, Rodrigo F.; GAVA, Vagner Luis; PESSÔA, Marcelo S.; SPINOLA, Mauro. M. **Uma proposta de processo de produção de aplicações web**. Revista Produção, vol. 15, n. 3, p. 376-389, Set./Dez. 2005.

HANSEN, Stuart A.; FOSSUM. Tim V. **Refactoring Model-View-Controller.** Journal of Computing Sciences in Colleges, vol. 21, no. 1, October 2005, 2005, p. 120-129.

LEFF, Avraham; RAYFIELD. James T. **Web-Application development using the modelniewlcontroller design pattern**. Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, p. 118-127. 2001.

MCCONNELL, Steve. **Code complete**. 2A ed., Microsoft Press, 2004.

MILANI, André. **MySQL – guia do programador**. São Paulo: Novatec, 2007.

MYSQL. **MySQL.** Disponível em: <http://www.mysql.com>. Acesso em: 29 mar. 2011.

NETBEANS. **NetBeans IDE**. Disponível em: <http://www.netbeans.org>. Acesso em: 22 jan. 2012.

PHP. **Linguagem PHP**. Disponível em: <http://www.php.net>. Acesso em: 05 mar. 2011.

PHPMYADMIN. **phpMyAdmin**. Disponível em: <http://www.phpmyadmin.net/home_page/index.php>. Acesso em: 11 fev. 2012.

PRESSMAN, Roger. **Engenharia de software**, 2005. Rio de Janeiro: McGrawHill.

SBC. **Sociedade Brasileira de Computação**. Disponível em: <http://www.sbc.org.br/index.php?option=com_content&view=category&layout=blog&id=303&Itemid=937>. Acesso em: 20 fev. 2012.

SWEAT, Jason E. **PHP architect's: guide to PHP design patterns**. Marco Tabini and Associates. Canada, 2005.

TIWARI, Anshuman; MITTAL, Kaushal. **Study of J2EE Patterns**. p. 1-12. Disponível em <http://www.it.iitb.ac.in/~kaushal/downloads/oos_project_report.pdf>. Acesso em: 05 jan. 2012.

TOMCAT. **Apache Tomcat**. Disponível em: <http://tomcat.apache.org/>. Acesso em: 15 fev. 2012.

WIKIPEDIA. **Olimpíada Brasileira de Informática**. Disponível em <http://pt.wikipedia.org/wiki/Olimp%C3%ADada_Brasileira_de_Inform%C3%A1tica>. Acesso em: 23 jan. 2012