

SUMÁRIO

APLICAÇÃO JOGO DO DINO UTILIZANDO PROGRAMAÇÃO	3
APLICAÇÃO DO FILTRO PREWITT PARA DETECÇÃO DE BORDAS EM IMAGENS	9
MULE VS APACHE CAMEL	16
BANCO DE DADOS ORIENTADO A OBJETOS PARA APLICAÇÕES ANDROID – ORMLite	22
SISTEMA DE IRRIGAÇÃO AUTOMATIZADO COM INTERFACE DE CONTROLE E MONITORAMENTO ONLINE	29
COMPARATIVO DO USO DE LINGUAGENS DE PROGRAMAÇÃO E GERADORES DE CÓDIGO NO DESENVOLVIMENTO DE SISTEMAS.....	36
DOCKER – APRESENTAÇÃO DA FERRAMENTA	46
PLAYER MR: REPRODUTOR DE ÁUDIO COM LEGENDAS	57
JOGO DAS DERIVADAS	63
UTILIZAÇÃO DOS SISTEMAS DE SINGLE SIGN-ON (SSO) DO GOOGLE E FACEBOOK	71
COMPARATIVO ENTRE BANCOS DE DADOS NÃO RELACIONAIS – NOSQL.....	78

Ficha

C749 Congresso Catarinense de Ciência da Computação
(4. : 2017 : Rio do Sul, SC).
Anais do IV Congresso Catarinense de Ciência da Computação. Rio do Sul, Santa Catarina, Junho 12-14; 20-21, 2017. / Organizador: Wesley dos Reis Bezerra. - Rio do Sul, SC, 2017.

Evento realizado pelo: Instituto Federal de Educação, Ciência e Tecnologia Catarinense - Campus Rio do Sul - SC.
1 CD-ROM

ISSN: 2319-0655.

1. Ciência da computação. 2. Tecnologia da informação e comunicação. 3. Ciência da Computação (Congressos). I. Bezerra, Wesley dos Reis. II. Instituto Federal de Educação, Ciência e Tecnologia Catarinense. III. Título.

CDD 23. ed. – 004

catalográfica elaborada pela Bibliotecária Caroline da Rosa Ferreira Becker
CRB14/807.

APLICAÇÃO JOGO DO DINO UTILIZANDO PROGRAMAÇÃO CONCORRENTE

Daryan Avi¹, Rodrigo Curvello²

¹Aluno Bacharelado em Ciência da Computação – IFC Rio do Sul

²Professor Bacharelado em Ciência da Computação – IFC Rio do Sul

daryan.spfc@gmail.com, rodrigo.curvello@ifc.edu.br

Abstract. *This article describes the development of the Dino Game application, which is inspired by Chrome Dino, a well-known game from the Google Chrome browser. It focuses on demonstrating the use of concurrent programming elements as well as their effectiveness in developing complex applications. This application was developed in the Java programming language through the Eclipse IDE. The overall goal was to develop a "Chrome Dino for desktop", but inserting new elements for the gameplay and having a greater variety of graphic elements, making it necessary to use multiple threads for its development.*

Key-words: *Jogo do Dino; Chrome Dino; Concurrent Programming.*

Resumo. *O presente artigo descreve o desenvolvimento da aplicação Jogo do Dino, a qual é inspirada no Chrome Dino, um conhecido jogo do navegador Google Chrome. Tem como foco demonstrar a utilização de elementos de programação concorrente, bem como sua eficácia para desenvolver aplicações complexas. Esta aplicação foi desenvolvida na linguagem de programação Java, por meio da IDE Eclipse. O objetivo geral foi desenvolver um "Chrome Dino para desktop", porém inserindo novos elementos para a jogabilidade e contando com maior variedade de elementos gráficos, fazendo-se necessária a utilização de múltiplas threads para seu desenvolvimento.*

Palavras-chave: *Jogo do Dino; Chrome Dino; Programação Concorrente.*

1. Introdução

Programação concorrente é uma forma de programação em que diferentes cálculos e execuções de processos são realizados de forma simultânea. Este recurso é fundamental para criar aplicações mais complexas, como jogos (ALMASI; GOTTLIEB, 1989).

Neste contexto, foram utilizados diversos elementos de programação concorrente para o desenvolvimento do Jogo do Dino, um jogo baseado no Chrome Dino, famoso jogo do dinossauro disponível para o navegador Google Chrome (GOOGLE, 2017).

No Chrome Dino, o usuário controla um dinossauro que precisa pular por cima de cactos e desviar de pterossauros para se manter vivo e pontuar. O Jogo do Dino conta com o mesmo dinossauro e a mesma mecânica de pular por cima de cactos, porém

contém mais elementos que proporcionam uma jogabilidade mais avançada. No lugar da ambientação em preto e branco, há um cenário colorido com diversos elementos operando paralelamente. No lugar de pterossauros, há agora naves alienígenas que voam de tempos em tempos no cenário e atiram *lasers* que precisam ser desviados pelo usuário. O objetivo é criar um cenário mais “sinistro” para o jogo, satirizando a extinção dos dinossauros ao criar uma versão diferente para a história: em vez de um cometa, um ataque alienígena.

2. Linguagem de programação e bibliotecas utilizadas

Para o desenvolvimento do *game*, foi utilizada a linguagem de programação Java - que tem como uma de suas características o suporte a *multithread* -, por meio da IDE Eclipse. As telas e componentes gráficos externos ao jogo foram criados utilizando as bibliotecas nativas AWT e Swing. Para reproduzir os desenhos utilizados no jogo, foi utilizada a biblioteca externa OpenGL. A programação concorrente foi feita com auxílio da classe Thread. Para a persistência, foi utilizado a biblioteca IO.

3. Recursos de programação concorrente

Foram implementadas diversas *threads* no código para controlar diferentes elementos do jogo. A maioria destas *threads* foram implementadas na classe Controle, responsável por todo o gerenciamento dos métodos e instâncias de classes utilizadas na aplicação. Entre as *threads* utilizadas, estão:

- Principal: *thread* responsável pelo método ‘main’, que inicia a execução do programa.
- Animator: objeto que gera as imagens gráficas *frame a frame*, atualizando a tela constantemente.
- Cronômetro: um *timer* que gerencia a contagem de tempo no jogo, bem como a adição de alguns objetos gráficos.
- Cronômetro Pata: um *timer* que executa a cada 0.15 segundos, trocando a pata do dinossauro a ser desenhada, dando uma sensação de movimento ao personagem.
- Clip: *thread* responsável por executar a música de fundo.

A figura 1 contém o código utilizado na inicialização da *thread timer* (o cronômetro). A cada segundo (1000 milissegundos, como definido na linha 135) o *timer* executa um conjunto de ações, como atualizar o tempo (linha 140) e verificar se novos objetos devem ser lançados no jogo (linhas 144 e 147).

```

135 timer = new Timer(1000, new ActionListener()
136 {
137     @Override
138     public void actionPerformed(ActionEvent e)
139     {
140         tempo ++;
141
142         telaJogo.atualizarTempo();
143
144         if (tempo % TEMPO_CACTUS_BOOST == 0)
145             desenho.aumentarVelocidadeCactus();
146
147         if (tempo >= TEMPO_NAVE_MINIMO && tempo % TEMPO_NAVE_RESPAWN == 0)
148             desenho.addNave();
149     }
150 });
151 timer.start();

```

Figura 1 - Código demonstrando a inicialização da thread Cronômetro.

Todas as *threads* acima são executadas simultaneamente durante o jogo.

4. Comandos e instruções do jogo

O dinossauro pode pular ou se mover para os lados, usando as setas do teclado. Caso o tronco do dinossauro encoste em um cactus ou em um *laser* lançado pelas naves espaciais, o jogo termina. Um *timer* conta o tempo transcorrido na partida. O objetivo é conseguir sobreviver o máximo de tempo possível, tentando bater o recorde de tempo estabelecido.

A velocidade dos cactos aumenta levemente a cada 3 segundos, sem haver um limite estabelecido. As naves aparecem a cada 12 segundos a partir do segundo 24, lançando *lasers* de um lugar gerado aleatoriamente.

A figura 2 contém o código do controle dos pulos do dinossauro. Se o *status* atual dele for “pulando” a sua altura aumentará; se for descendo, diminuirá; se não for nenhum dos dois, se manterá.

```

153 if (status == STATUS_PULANDO && y < PULO_ALTURA)
154 {
155     y += PULO_ACRESCIMO;
156
157     if (y > PULO_ALTURA)
158         status = STATUS_DESCENDO;
159 }
160 else if (status == STATUS_DESCENDO && y > PADRAO_Y)
161 {
162     y = Math.max(PADRAO_Y, y - PULO_ACRESCIMO * 1.3f);
163 }
164 else
165 {
166     status = STATUS_SOLO;
167 }

```

Figura 2 - Parte da classe Dinossauro responsável pelo controle de altura.

5. Objetos gráficos

A classe Desenho implementa a interface GLEventListener, a qual contém os métodos de desenho, como o método display. Dentro desta classe são instanciados vários objetos que representam cada parte do desenho – Dinossauro, Céu, Nuvens, etc.

Todos esses objetos herdam da classe abstrata Objeto (figura 5). No construtor desta classe são passados parâmetros de coordenadas x e y, largura e altura. O método desenhar deverá ser sobrescrito pelos herdeiros desta classe.

```

12 public abstract class Objeto
13 {
14     public float x, y, largura, altura;
15
16     protected Objeto(float x, float y, float largura, float altura)
17     {
18         this.x = x;
19         this.y = y;
20         this.largura = largura;
21         this.altura = altura;
22     }
23
24     protected abstract void desenhar(GL2 gl2);
25 }

```

Figura 3 - Código da classe Objeto.

6. Persistência

A persistência é utilizada no jogo para a gravação do *high score* (melhor resultado) do jogador. A classe Recorde é a responsável por fazer a busca e a gravação do resultado em um arquivo de texto de nome 'Recorde.txt', sendo que é escrito apenas um número no arquivo: o total de segundos transcorridos durante a partida.

O método gravar é demonstrado na figura 3. Nele, é realizada a gravação com o auxílio da classe BufferedWriter, passando como parâmetro o resultado no método write (linha 79).

```

72 public static void gravar(int resultado)
73 {
74     BufferedWriter writer = null;
75
76     try
77     {
78         writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(CAMINHO), "utf-8"));
79         writer.write(Integer.toString(resultado));
80     }
81     catch (Exception e)
82     {
83         e.printStackTrace();
84     }
85     finally
86     {
87         if (writer != null)
88             try { writer.close(); } catch (Exception exc) {};
89
90         writer = null;
91     }
92 }

```

Figura 4 - Método responsável por gravar o resultado no arquivo.

7. Áudio

Para executar a música ao fundo, foi utilizada a classe `Clip`. Conforme mostrado na figura 4, a instância desta classe abre um *stream* de áudio no sistema e inicia sua execução.

```
175      AudioInputStream audio = AudioSystem.getAudioInputStream(new File("audio/Tererê.wav"));
176
177      clip = AudioSystem.getClip();
178      clip.open(audio);
179      clip.start();
```

Figura 5 - Instanciamento da classe `Clip`.

A música escolhida para o jogo foi ‘Tererê’, da banda catarinense Apicultores Clandestinos (APICULTORES CLANDESTINOS, 2015). A música é usada para dar um clima mais “sinistro” ao jogo, combinando com as naves extraterrestres que atacam o dinossauro.

8. Progresso do jogo passo a passo

Ao iniciar a aplicação, é chamado o método ‘configurações iniciais’ na classe `Controle`. Esta classe gerenciará todo o funcionamento da aplicação.

Em seguida, é instanciada a classe `TelaCarregamento`, um *frame* que executará enquanto as classes gráficas são instanciadas.

Ao fim do carregamento, a classe `Controle` se encarregará de instanciar todas as *threads*, a classe `Desenho` e as janelas. É instanciada a classe `TelaPrincipal`, o *frame* que contém toda a interface gráfica do jogo. Sobre a tela principal é aberta a classe `TelaOpcoes`, que herda da classe `JDialog`.

Finalmente, quando é acionado o botão `Iniciar`, a tela principal ganha foco e o jogo se inicia.



Figura 6 - Jogo do Dino.

Quando o jogo termina, novamente é instanciada a Tela de Opções, agora com mensagem a “Game Over” e congratulações se o recorde de tempo tiver sido batido.



Figura 7 - Tela de Opções (término).

9. Diagrama de classes

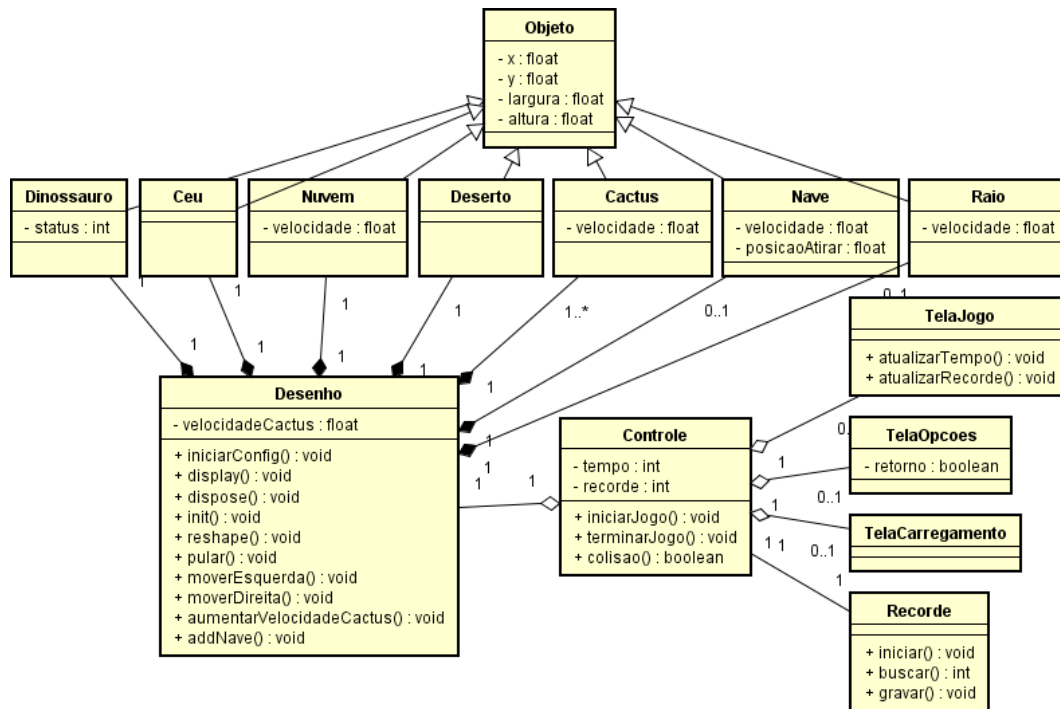


Figura 8 - Diagrama de classes.

10. Conclusão

No geral, o Jogo do Dino mostrou-se eficiente e executou sem travamentos, mesmo com a utilização de várias *threads* operando concorrentemente. Contudo, o carregamento de recursos da biblioteca OpenGL mostrou-se relativamente lento, fazendo com que o usuário precisa esperar alguns segundos antes de poder jogar. Uma otimização nestes recursos deve ser pensada para aplicações futuras.

O desenvolvimento de cada parte do desenho foi feita de forma manual, o que levou a um código limpo e otimizado, mas também a uma grande quantidade de tempo programando. Fica claro que a utilização da biblioteca OpenGL de forma “rústica” serve para aplicações gráficas pequenas, mas se torna ineficiente conforme a complexidade aumenta.

Referências

ALMASI, G.S. e A. GOTTLIEB (1989). **Highly Parallel Computing**. Benjamin-Cummings, Redwood City, CA.

APICULTORES CLANDESTINOS (2015). **Tererê**. Disponível em:

<https://apicultoresclandestinos.bandcamp.com/track/terer>. Acesso: Março de 2017.

GOOGLE, Inc. (2017). **Chrome Dino Game**. Disponível em: <https://chromedino.com/>. Acesso: Março de 2017.

APLICAÇÃO DO FILTRO PREWITT PARA DETECÇÃO DE BORDAS EM IMAGENS

Daryan Avi, André Alessandro Stein

Aluno Bacharelado em Ciência da Computação – IFC Rio do Sul

Professor Bacharelado em Ciência da Computação – IFC Rio do Sul

daryan.spfc@gmail.com, andre.stein@ifc.edu.br

Abstract. *This article aims to show the concept of the Prewitt filter for edge detection in images, as well as its practical application. For the application, an algorithm was developed in the Java language, through the Eclipse IDE. The algorithm contains all the mathematical formulas used in the filter, and applies them to bitmap images. The article demonstrates the results of applying the filter over some images and analyzes its performance and effectiveness.*

Key-words: Filter; Prewitt; Algorithm.

Resumo. *Este artigo tem como objetivo mostrar o conceito do filtro de Prewitt para detecção de bordas em imagens, bem como sua aplicação prática. Para a aplicação, foi desenvolvido um algoritmo na linguagem Java, por meio da IDE Eclipse. O algoritmo contém todas as fórmulas matemáticas utilizadas no filtro, e as aplica sobre imagens do tipo bitmap. O artigo demonstra o resultado da aplicação do filtro em cima de algumas imagens e analisa o seu desempenho e eficácia.*

Palavras-chave: Filtro; Prewitt; Algoritmo.

1. Introdução

Em computação gráfica, a detecção de bordas inclui uma variedade de métodos matemáticos utilizados com o objetivo de identificar pontos numa imagem digital em que o brilho muda bruscamente ou, de maneira mais formal, tem descontinuidades. A detecção de bordas é uma ferramenta fundamental para o processamento de imagens (UMBAUGH, 2010).

Uma das formas de realizar a detecção de bordas é através da filtragem de passa-alta. Filtros de passa-alta ou de acentuação têm como características eliminar baixas frequências, atenuar altas frequências, realçar o contraste e os detalhes em imagens e destacar características como bordas, linhas, curvas e manchas. Na filtragem de passa-alta, os componentes de alta frequência da transformada de Fourier não são alterados, enquanto que os de baixa frequência são removidos. Isto faz com que os detalhes finos da imagem sejam enfatizados (IC UFF, 2017).

O objetivo deste artigo será estudar o desenvolvimento e os resultados da aplicação de um dos filtros de passa-alta: o filtro de Prewitt.

2. Descrição do filtro

O filtro ou operador de Prewitt é utilizado particularmente em algoritmos de detecção de borda. Tecnicamente, trata-se de um operador de diferença, computando uma aproximação do gradiente da função de intensificação da imagem. Em cada ponto da imagem, o resultado do filtro de Prewitt é ou o vetor gradiente correspondente, ou a norma do vetor. O filtro de Prewitt baseia-se em convolver a imagem com uma filtragem pequena e separável nas direções vertical e horizontal, e, portanto, tem um custo computacional relativamente baixo. Por outro lado, a filtragem que ele produz é relativamente “bruta”, em particular quando há uma alta frequência de variações na imagem (PREWITT, 1970).

Matematicamente, o filtro usa duas máscaras 3 x 3 que são convolidas com a imagem original para calcular aproximações das derivadas – uma para mudanças horizontais e uma para verticais (PREWITT, 1970).

Sendo A a imagem original, a máscara utilizada para o eixo horizontal, G_x, acentua os valores na coluna à esquerda do *pixel* e diminui os valores na coluna à direita, como mostra a figura 1.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

Figura 9 - Máscara utilizada no eixo horizontal

Já a máscara utilizada para o eixo vertical, G_y, acentua os valores na linha cima do *pixel* e diminui os valores na linha abaixo, como mostra a figura 2.

$$\mathbf{G}_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$

Figura 10 - Máscara utilizada no eixo vertical

Em cada ponto da imagem, as aproximações de gradiente resultantes podem ser combinadas para dar a magnitude do gradiente, calculando a raiz quadrada do quadrado de G_x mais o quadrado de G_y, como mostra a figura 3.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

Figura 11 - Fórmula da combinação dos gradientes resultantes

3. Metodologia

A operação de filtragem foi feita em cima de imagens do tipo *bitmap*. Imagens *bitmap* são imagens “pixelizadas”, ou seja, um conjunto de pontos (*pixels*) contidos num quadro, onde cada um deles possui um ou mais valores que descrevem uma cor (CCM, 2017).

A aplicação que faz a filtragem foi desenvolvida na linguagem Java, utilizando a IDE Eclipse.

3.1. Algoritmo

O modelo possui três classes principais: PBMImage, PGMImage e PPMImage, cada uma sendo a representação de um tipo de imagem *bitmap* na forma de um objeto. Cada classe faz a leitura e gravação de arquivos de acordo com seu respectivo tipo, guardando dados como altura, largura e valores de cada *pixel* da imagem.

O método que faz todo o processamento do filtro de Prewitt consta na classe PGMImage, e é mostrado na figura 4.

```

674 public PBMImage prewittOperator(int threshold)
675 {
676     ArrayList<ArrayList<Integer>> newLines = new ArrayList<>();
677     ArrayList<Integer> newLine;
678
679     for (int y = 0; y < height; y++)
680     {
681         newLine = new ArrayList<>();
682
683         for (int x = 0; x < width; x++)
684         {
685             int yUp = Math.max(y - 1, 0);
686             int yDown = Math.min(y + 1, height - 1);
687             int xLeft = Math.max(x - 1, 0);
688             int xRight = Math.min(x + 1, width - 1);
689
690             int[][] v = new int[3][3]
691             {
692                 {lines.get(yUp).get(xLeft), lines.get(yUp).get(x), lines.get(yUp).get(xRight)},
693                 {lines.get(y).get(xLeft), 0, lines.get(y).get(xRight)},
694                 {lines.get(yDown).get(xLeft), lines.get(yDown).get(x), lines.get(yDown).get(xRight)}
695             };
696
697             int gx = (v[0][0] + v[1][0] + v[2][0]) - (v[0][2] + v[1][2] + v[2][2]);
698             int gy = (v[0][0] + v[0][1] + v[0][2]) - (v[2][0] + v[2][1] + v[2][2]);
699
700             int edge = (int) Math.sqrt((gx * gx) + (gy * gy));
701
702             newLine.add(edge < 0 ? 0 : edge > colors ? colors : edge);
703         }
704         newLines.add(newLine);
705     }
706
707     return new PGMImage(width, height, newLines, colors).toPBM(threshold);
708 }
709

```

Figura 12 - Código do método do filtro de Prewitt

Como mostra a figura acima, na chamada do método, é passado como parâmetro o valor do limiar de conversão da imagem (*threshold*) de tom de cinza para preto e branco; este limiar definirá o quão acentuada será a detecção de bordas na imagem.

É feita a chamada de um *for* que percorrerá todas as linhas da imagem, de acordo com sua altura (conforme linha 679). Dentro de cada linha, há um *for* interno que percorre cada coluna da imagem, de acordo com sua largura (linha, 683). Na linha 690, é criada uma matriz 3 x 3 representando o ponto atual na execução junto aos 8 *pixels* que estão ao seu redor.

A variável *gx* guardará o valor da máscara horizontal, fazendo o cálculo conforme mostrado na figura 1. A variável *gy* guardará o valor da máscara vertical, fazendo o cálculo conforme mostrado na figura 2. Por fim, é calculada a borda pela variável *edge* (conforme figura 3) e guardada na lista de *pixels*.

Quando o programa executar estas operações para todos os *pixels* da imagem, é gerada uma imagem PGM com os valores armazenados (linha 708). Esta imagem será convertida para o formato PBM utilizando o método *toPBM()*, passando como parâmetro o limiar de conversão (variável *threshold*). Por fim, esta imagem do tipo PBM será salva no sistema conforme definido pelo usuário.

3.2. Interface gráfica

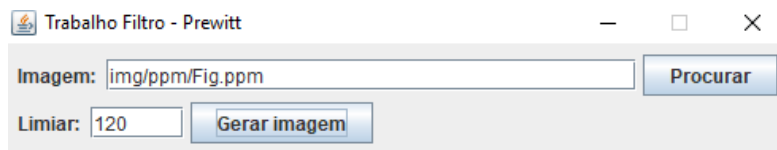


Figura 13 - Interface da aplicação

A figura 5 mostra a interface do programa. No campo “Imagem” é passado como parâmetro o local do arquivo da imagem, que deve estar no formato PBM, PGM ou PPM. Se a imagem for no formato PBM ou PPM, o programa faz a conversão para PGM.

No campo “Limiar”, é definido o limiar para a filtragem da imagem. Quanto mais próximo o limiar estiver de 0, mais acentuadas serão as bordas na imagem.

Ao clicar em “Gerar imagem”, a imagem escolhida será filtrada pelo método *prewittOperator()*, passando como parâmetro o limiar escolhido, e o resultado será salvo no local de escolha do usuário.

4. Resultados

Para testar a aplicação, foi utilizada a imagem colorida de uma mulher de chapéu, no formato PPM. A imagem é mostrada na figura 6.



Figura 14 - Imagem utilizada para os testes da aplicação

Foram feitos três testes principais com a imagem: um utilizando um limiar de conversão de 64, um utilizando 128 e outro 192.

A figura 7 mostra o resultado utilizando o limiar de 64.



Figura 15 - Imagem filtrada utilizando um limiar de conversão de 64

A figura 8 mostra o resultado utilizando o limiar de 128.



Figura 16 - Imagem filtrada utilizando um limiar de conversão de 128

Por fim, a figura 8 mostra o resultado utilizando o limiar de 192.



Figura 17 - Imagem filtrada utilizando um limiar de conversão de 192

Os resultados mostram que o filtro separa mais bordas quanto mais baixo for o limiar de conversão. Como mostrado na figura 7, utilizando um limiar de conversão de 64, as bordas na imagem são bastante acentuadas, mas também há mais ruído do que o desejado. Conforme o limiar de conversão aumenta, há menos bordas e menos ruído. Enquanto na figura 7 são mostrados muitos detalhes no rosto e no chapéu da mulher, na figura 8 os detalhes ficam mais difíceis de enxergar, e na figura 9 são quase indistinguíveis, mostrando apenas alguns poucos contornos no rosto da mulher.

5. Conclusão

O objetivo geral do artigo foi alcançado. O algoritmo foi capaz de fazer o processamento e filtragem da imagem em um pequeno tempo de execução, mesmo utilizando imagens de tamanho considerável. Por se tratar da utilização de máscaras simples em cima de cada ponto da imagem, o filtro de Prewitt não tem um alto custo computacional.

Os resultados dos testes mostraram que o filtro tem capacidade de detecção de bordas limitada em relação a algoritmos mais complexos. Utilizando um limiar de conversão menor, as imagens resultantes têm suas bordas mais acentuadas, mas também contém uma quantidade de ruído maior do que o desejado.

Conclui-se que o filtro é simples, eficiente e de fácil utilização. Contudo, pode não ser tão efetivo quando utilizado para detecção de bordas em imagens muito detalhadas, sendo necessária a utilização de algoritmos mais complexos em seu lugar.

Referências

CCM. **Imagens bitmap e vetoriais**. Disponível em: <<http://br.ccm.net/contents/737-imagens-bitmap-e-vetoriais>>. Consulta: Maio de 2017.

IC UFF, Instituto de Computação – Universidade Federal Fluminense. **Computação Gráfica – Filtragem de Imagens**. v. 2, c.5, 2017.

PREWITT, J.M.S. "Object Enhancement and Extraction" in "Picture processing and Psychopictorics". Academic Press, 1970.

UMBAUGH, Scott E. **Digital image processing and analysis : human and computer vision applications with CVIPtools** (2nd ed.). Boca Raton, FL: CRC Press, 2010.

MULE VS APACHE CAMEL

Daryan Avi¹, Wesley dos Reis Bezerra²

¹Aluno Bacharelado em Ciência da Computação – IFC Rio do Sul

²Professor Bacharelado em Ciência da Computação – IFC Rio do Sul

daryan.spfc@gmail.com, wesley.bezerra@ifc.edu.br

Abstract. *Enterprise Service Buses (ESBs) are distributed systems tools that implement a system of communication between software applications that interact mutually in a service-oriented architecture. This article aims to explain the operation of ESBs and to compare two of the most popular ESB tools: Mule and Apache Camel. To make this comparison, elements such as flexibility, ease of learning, graphical interface and scalability are analyzed.*

Key-words: ESB; Mule; Camel.

Resumo. *Os Enterprise Service Bus (ESBs) são ferramentas de sistemas distribuídos que implementam um sistema de comunicação entre aplicativos de software que interagem mutuamente em uma arquitetura orientada a serviços. Este artigo tem como objetivo explicar o funcionamento dos ESBs e fazer um comparativo entre duas das ferramentas ESB mais populares: Mule e Apache Camel. Para fazer a comparação, são analisados elementos como flexibilidade, facilidade de aprendizado, interface gráfica e escalabilidade.*

Palavras-chave: ESB; Mule; Camel.

1. Introdução

Um Enterprise Service Bus (ESB) permite implementar um sistema de comunicação entre aplicativos de *software* que interagem mutuamente em uma arquitetura orientada a serviços (SOA). Como implementa uma arquitetura de *software* para computação distribuída, também implementa uma variante especial do modelo cliente-servidor mais geral. O ESB promove agilidade e flexibilidade em relação à comunicação de alto nível de protocolo entre aplicações (CHAPPELL, 2004).

Projetos de integração de aplicativos requerem visão operacional, pensamento estratégico e muita diligência. Mesmo assim, pode ser difícil determinar qual oferta de integração melhor se adequa às suas necessidades de negócios. Uma das tarefas mais difíceis pode ser entender como as soluções concorrentes se comparam (CHAPPELL, 2004).

Para ajudar a superar estes desafios, este artigo compara duas soluções de integração populares, Mule ESB e Apache Camel.

2. Visão geral do ESB

O conceito de ESB é análogo ao conceito de barramento encontrado na arquitetura de *hardware* do computador combinado com o *design* modular e concorrente de sistemas operacionais de computadores de alto desempenho. A motivação foi encontrar um conceito padrão, estruturado e de propósito geral para descrever a implementação de componentes de *software* acoplados de forma frouxa (chamados de serviços) que devem ser implementados, executados, heterogêneos e díspares de forma independente dentro de uma rede. ESB é também um padrão de implementação comum para a arquitetura orientada a serviços (BELL, 2008).

Um ESB transporta o conceito de *design* de sistemas operacionais modernos para redes de computadores distintos e independentes. Como sistemas operacionais concorrentes, um ESB atende a serviços de commodities, além de adoção, tradução e encaminhamento de uma solicitação de cliente para o serviço de atendimento apropriado (CHRISTUDAS, 2008).

Segundo CHRISTUDAS (2008), as funções primárias de um ESB são:

- Monitorar e controlar o encaminhamento da troca de mensagens entre serviços;
- Resolver contenção entre componentes de serviço de comunicação;
- Controle de implantação e controle de versão de serviços;
- Dispor o uso de serviços redundantes;
- Atende a serviços de commodities como manipulação de eventos, transformação e mapeamento de dados, filas de mensagens e eventos e sequenciamento, manipulação de segurança ou exceção, conversão de protocolos e aplicação da qualidade adequada do serviço de comunicação.

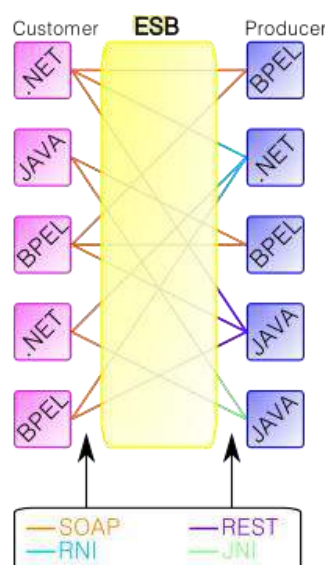


Figura 18 - Modelo ESB Produtor-Consumidor

Como mostra a figura 1, todos os serviços ao consumidor se comunicam da mesma forma com o ESB: o ESB traduz uma mensagem para o tipo de mensagem correta e envia a mensagem para o serviço de produtor correto.

3. Mule

O Mule é um ESB leve e baseado em Java, que permite aos desenvolvedores conectar aplicativos de forma rápida e fácil, permitindo que eles troquem dados. Permite integração de sistemas existentes, independentemente das diferentes tecnologias que as aplicações utilizam, incluindo JMS, Web Services, JDBC, HTTP e muito mais. O ESB pode ser implantado em qualquer lugar, pode integrar e orquestrar eventos em tempo real ou em lote, e tem conectividade universal (MULE ESB, 2017).

Mule tem capacidades que incluem:

- Criação e hospedagem de serviços – expor e hospedar serviços reutilizáveis, usando o ESB como um *container* de serviço leve.
- Mediação de serviços – protege os serviços de formatos e protocolos de mensagens, separa a lógica de negócios de mensagens e habilita chamadas de serviço independentes da localização.
- Roteamento de mensagens – rotear, filtrar, agregar e re-sequenciar mensagens com base no conteúdo e regras.
- Transformação de dados – troca de dados entre diferentes formatos e protocolos de transporte.

4. Apache Camel

O Apache Camel é um mecanismo de roteamento e mediação baseado em regras, similar a um ESB, que fornece uma implementação baseada em objetos Java dos Padrões de Integração Empresarial usando uma *interface* de programação de aplicativo para configurar roteamento e regras de mediação (IBSEN, 2010).

Camel dá a capacidade ao usuário de definir regras de roteamento e mediação em uma variedade de linguagens específicas de domínio, incluindo arquivos de configuração XML Spring ou Blueprint e um Scala DSL. Isso significa que o usuário obtém a conclusão inteligente das regras de roteamento no IDE, seja em um editor Java, Scala ou XML.

O Apache Camel usa URIs para trabalhar diretamente com qualquer tipo de modelo de transporte ou de mensagens, como HTTP, ActiveMQ, JMS, JBI, SCA, MINA ou CXF, bem como opções de Componentes e Formato de Dados conectáveis. Apache Camel é uma pequena biblioteca com dependências mínimas para fácil incorporação em qualquer aplicativo Java. O Apache Camel permite trabalhar com a mesma API, independentemente do tipo de transporte utilizado.

Apache Camel oferece suporte para Bean Binding e integração com *frameworks* populares como CDI, Spring, Blueprint e Guice. Camel também tem suporte extensivo para teste de unidade de suas rotas (APACHE CAMEL, 2017).

5. Comparação: Mule vs Apache Camel

Em primeiro lugar, esses dois produtos compartilham várias propriedades importantes, o que os torna bem-sucedidos em projetos básicos de integração:

- Código aberto: Mule e Camel são produtos *open-source* que cumprem os padrões globais e fornecem código fonte facilmente disponível.
- Leve: Essas soluções são leves – os tamanhos de *download* não ultrapassam 80MB – e ambos são conhecidos por seu baixo tamanho e interoperabilidade com outras tecnologias de código aberto e ferramentas de desenvolvimento.
- Comunidades Ativas: Uma comunidade online ativa de desenvolvedores está presente em ambos os produtos (ou seja, Camel tem aproximadamente 18.000 usuários em fóruns Mule tem mais de 100.000); Essas comunidades fornecem *feedback*, documentação e tutoriais.

Essas qualidades compartilhadas fornecem uma base sólida para ambas as soluções; no entanto, sua abordagem de integração difere, e por sua vez, define sua força como uma solução empresarial.

Embora conectar dois aplicativos simples pode exigir pouco mais do que a integração ponto-a-ponto básica, a empresa de hoje requer uma solução de integração flexível, fácil de usar e escalável para crescimento futuro. Abaixo, são comparadas ambas as soluções com base nos seguintes critérios: flexibilidade, facilidade de aprendizado, *interface* gráfica e escalabilidade.

5.1. Flexibilidade

O Mule ESB é uma plataforma de integração completa e, como tal, possui um *container* de serviço e uma estrutura de mediação como parte da oferta. Isso permite que a Mule atenda aos requisitos não funcionais, como confiabilidade, alta disponibilidade, escalabilidade e segurança corporativa, além dos requisitos funcionais esperados. Se um *container* de serviço estiver além do escopo do seu projeto, o Mule pode ser reduzido e usado como uma estrutura de integração incorporada. Essa flexibilidade permite que as empresas adaptem Mule a projetos de integração específicos.

Camel, por si só, é uma estrutura de mediação e não pode fornecer muitos dos requisitos essenciais não-funcionais de uma solução de integração. Para superar esse obstáculo, os provedores de integração frequentemente combinam Camel com Apache ServiceMix ou outros recipientes de serviço de terceiros. Esta abordagem de *bricolage* para criar uma plataforma de integração completa adiciona trabalho adicional para desenvolvedores e aumenta o risco de criar uma plataforma quebradiça.

5.2. Facilidade no aprendizado

O *framework* enxuto do Camel torna seu aprendizado mais fácil para os programadores. Camel também acomoda diferentes Linguagens Específicas de Domínio (DSLs), permitindo que os programadores trabalhem em qualquer idioma que acharem mais confortável. Camel também fecha a lacuna entre a modelagem e a implementação aderindo aos padrões de integração empresarial (EIPs) - permitindo aos programadores dividir os problemas de integração em partes menores que são mais facilmente compreendidas.

Mule também é fácil de aprender, e fornece ambos DSLs múltiplos ao aderir aos EIPs. Além disso, o ambiente de desenvolvimento visual da Mule torna as tarefas historicamente difíceis, como o mapeamento de dados, fácil de aprender, fornecendo funcionalidade de arrastar e soltar. Ambas as empresas oferecem a TI empresarial uma solução altamente utilizável que torna os projetos de integração maiores mais gerenciáveis. Isso se torna cada vez mais importante à medida que as organizações se expandem, e eles precisam contratar recursos técnicos adicionais.

5.3. Interface gráfica

Mule fornece um ambiente de desenvolvimento visual intuitivo, o Anypoint Studio. Esse ambiente baseado em Eclipse fornece funcionalidade de arrastar e soltar e permite que os desenvolvedores se concentrem em conceitos de alto nível em vez de detalhes técnicos. O Anypoint Studio é fácil de aprender e torna os desenvolvedores mais produtivos, mais rapidamente. Com o ambiente de *design* gráfico desta ferramenta, mesmo recursos não técnicos podem executar tarefas historicamente difíceis

Camel, no entanto, não tem um ambiente de desenvolvimento visual e sem ferramentas adicionais é apenas eficiente para um público limitado. À medida que os pontos de integração crescem, o quadro de Camel se torna menos gerenciável, a maioria das empresas optará por adicionar um ambiente de desenvolvimento visual ao Camel.

5.4. Escalabilidade

O Mule tem conectividade instantânea da API com centenas de aplicações e serviços baseados na nuvem mais populares. Esses conectores também integram *interfaces* proprietárias como SAP, TIBCO Rendezvous, Oracle Siebel CRM, PayPal ou CICS Transaction Gateway da IBM. Mule também fornece um *kit* de desenvolvimento que permite aos desenvolvedores usar anotações Java para construir rapidamente extensões Mule que se integram diretamente com o Anypoint Studio.

Camel tem dezenas de transportes, não centenas. Camel também não tem um SDK definido e as empresas são deixadas para usar *kits* de terceiros ou forçadas a esperar por um fornecedor comercial para adicionar suporte para a sua API ou conector. Ao contrário de Mule, Camel não fornece integração de aplicativos empacotados, e as empresas são obrigadas a realizar esses projetos “em casa”.

6. Conclusão

Enquanto Mule e Camel oferecem soluções leves, cada um tem uma abordagem distinta à integração. O Mule ESB é uma plataforma pronta para usar com componentes estrategicamente alinhados e é mais adequada para empresas que necessitem de um ESB ágil com um tempo de lançamento rápido. Por outro lado, Camel optou por fornecer um *framework* “esqueleto”, colocando a carga técnica sobre a empresa para construir uma plataforma funcional. Esta abordagem pode apelar para o desejo de um desenvolvedor de construir e alterar, mas rotineiramente causa atrasos

inesperados, perda de funcionalidade e puxa a atenção para longe das necessidades essenciais do negócio.

Em parte alguma isso é mais evidente do que na abordagem das linguagens de expressão nas duas soluções. Mule define uma linguagem de expressão preferida, enquanto ainda permite que os desenvolvedores usem uma ampla lista de outras. Este caminho pretendido remove o tempo de pesquisa necessário para determinar qual linguagem de expressão funcionará melhor – enquanto ainda fornece alternativas. No entanto, a abordagem “sem as mãos” do Camel simplesmente oferece a capacidade de usar uma série de linguagens de expressão, mas não fornece orientação ou prospectiva estratégica.

Referências

APACHE CAMEL. **Apache Camel**. Disponível em: <<http://camel.apache.org/>>. Acesso em 22 de maio de 2017.

BELL, Michael. **Service-Oriented Modeling: Service Analysis, Design, and Architecture**. Wiley & Sons, 2008.

CHAPPELL, David. **Enterprise Service Bus**. O'Reilly, 2004.

CHRISTUDAS, Binildas A. **Service-oriented Java Business Integration**. Packt Publishers, 2008.

IBSEN, Claus; ANSTEY, Jonathan. **Camel in Action**. Manning Publications, 2010.

MULE ESB. **What is Mule ESB?** Disponível em: <<https://www.mulesoft.com/resources/esb/what-mule-esb>>. Acesso em 22 de maio de 2017.

Banco de Dados Orientado a Objetos para Aplicações Android – ORMLite

Alex Manoel Coelho¹, Cristhian Heck²

¹Aluno 7ª fase do Curso de Ciência da Computação do Instituto Federal Catarinense –
Campus Rio do Sul

²Professor de Ensino Superior do Instituto Federal Catarinense – Campus Rio do Sul
alexma_coelho@hotmail.com.br, cristhian.heck@ifc-riodosul.edu.br

Abstract. *This meta-article describes the use of the ORMLite object-oriented database, which is a way to persist data in a mobile application, created specifically for this type of development. You will also be approached the necessary settings for the operation of this database, in a mobile application, in this article, specifically on Android. The example that will be used to exemplify the use of ORMLite is very simple, aiming to show how to work with it in a mobile application, and what its advantages.*

Key-words: *ORMLite; Object-oriented data bank; Android.*

Resumo. *Este meta artigo descreve a utilização do banco de dados orientado a objeto ORMLite, sendo ele uma forma de persistir dados em uma aplicação mobile, criado especificamente para este tipo de desenvolvimento. Também será abordado as configurações necessárias para o funcionamento deste banco de dados, em uma aplicação mobile, neste artigo, especificamente no Android. O exemplo que será usado para exemplificar o uso do ORMLite é bem simples, tendo o objetivo de mostrar como trabalhar com ele em uma aplicação mobile, e quais as suas vantagens.*

Palavras-chave: *ORMLite; Banco de dados orientado a objetos; Android.*

1. Introdução

Segundo Elmasri e Navathe (2010), os Bancos de Dados possuem as seguintes propriedades explícitas: (1) Um banco de dados representa determinados aspectos do mundo real, às vezes sendo chamado de minimundo, onde as mudanças no minimundo são refletidas em um banco de dados; (2) Um banco de dados é uma coleção lógica e coesa de dados com algum significado inerente; (3) Um banco de dados é projetado, construído e povoado por dados, acatando a uma proposta específica.

Um Banco de Dados Relacional (BDR) é um banco de dados que modela os dados de uma forma que eles sejam percebidos pelo usuário como tabelas relacionadas (Elmasri e Navathe, 2011). Enquanto um Banco de Dados Orientado a Objetos (BDOO) pode ser utilizado como alternativa aos Bancos de Dados Relacionais para armazenar objetos compartilhados entre diferentes aplicações, eles se tornaram conhecidos com o crescente uso de linguagens orientada a objetos (Mansueli, 2016).

Os BDOOs partem de uma argumentação bem simples, onde o que se persiste são os objetos e, assim sendo, o seu “estado” é representado pelos atributos. Os atributos seriam equivalentes aos campos de uma tabela, enquanto as associações entre objetos podem ser comparadas aos relacionamentos em Sistemas Gerenciadores de Bancos de Dados Relacionais, criados como restrições de integridade referencial. (Mansueli 2016).

Quando se trata de armazenamento de dados para o sistema operacional Android, é comum a adoção do SQLite, um banco de dados relacional gratuito, livre, portátil e confiável (SQLite 2017). Mas, uma das dificuldades envolvendo bancos de dados relacionais e aplicações orientadas a objetos, é que suas classes não podem ser inseridas de uma forma direta no banco de dados, tendo a necessidade de definir um mapeamento entre cada classe, e a tabela correspondente no banco de dados. Ainda que a grande parte das aplicações para Android utilize o banco de dados relacional para persistência de dados, é possível adotar uma outra alternativa, optando por bancos de dados orientados a objetos.

O presente artigo tem como objetivo esclarecer como utilizar o *framework* ORMLite, que é um dos vários bancos de dados orientados a objetos. O motivo da escolha, é porque facilita a implementação do software e também de acordo com Neto, Lemos e Castro (2017) uma aplicação BDOO, tende a apresentar na maioria dos casos, um desempenho melhor, quando comparado a um banco de dados relacional. Além disso, como será visto nos tópicos a seguir, ele facilita e agiliza o desenvolvimento de uma aplicação interligada a um banco de dados.

2. Metodologia

Para o desenvolvimento da aplicação Android, será utilizado a IDE Android Studio, a qual automatiza e facilita muitas operações na criação de um aplicativo mobile para a plataforma Android. Nas seções a seguir, será mostrado o passo a passo de como configurar o ORMLite em aplicação para Android.

2.1 Adicionando ORMLite ao projeto

Para adicionar o ORMLite é necessário importar duas bibliotecas ao projeto, de forma que as dependências do *framework* possam ser adicionadas. Para isso é preciso seguir os seguintes passos: Clicar com o botão direito no projeto – Open Module Settings – Dependecies – Adicionar as bibliotecas: “ormlite-android” e “ormlite-core”. Essas duas bibliotecas são encontradas no site do ORMLite. Na Figura 1 é possível ver onde elas estarão localizadas.

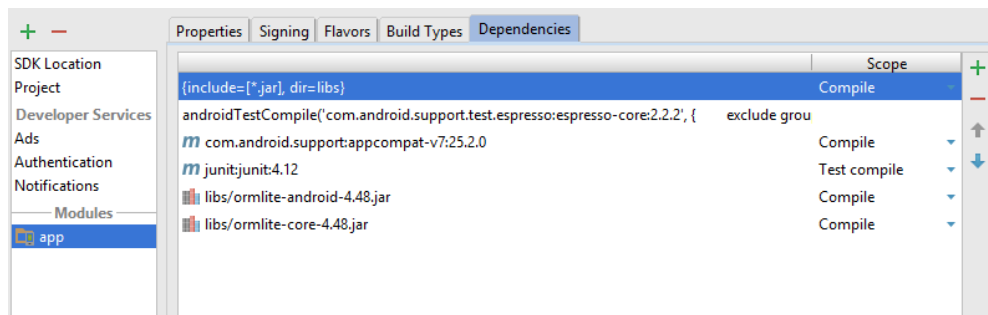


Figura 19 - Adicionando as bibliotecas

2.2 Persistindo uma classe

Para indicar quais classes que deverão ser persistidas, é preciso colocar as *notations*, desta forma indicando para o ORMLite, o que é uma tabela, o que é campo, o que é uma chave estrangeira, e assim por diante. Na figura 2, é possível analisar um exemplo do uso das *notations*.



```

4
5  @DatabaseTable(tableName = "disciplina")
6  public class Disciplina {
7      @DatabaseField(generatedId = true) //id auto increment
8      private long id;
9      @DatabaseField
10     private String nome;
11     @DatabaseField
12     private String codigo;
13     //chave estrangeira isso precisa, dessa forma a relação entre as tabelas prevalecerá
14     @DatabaseField(foreign = true)
15     private Estudante estudante;
16
17     public Disciplina() {
18     }
19
20     public Disciplina(String nome, String codigo) {
21         this.nome = nome;
22         this.codigo = codigo;
23     }
24
25     public long getId() { return id; }
26
27     public void setId(long id) { this.id = id; }
28
29     public String getNome() { return nome; }
30
31     public void setNome(String nome) { this.nome = nome; }
32
33     public String getCodigo() { return codigo; }
34
35     public void setCodigo(String codigo) { this.codigo = codigo; }

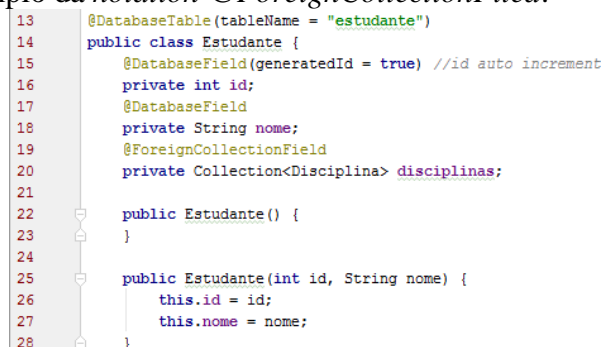
```

Figura 20 - Usando Notations para persistir a classe Disciplina

Analisando o código, na linha 5 é definido o nome que será dado a tabela, neste caso, “disciplina”. Na linha 7 está indicando o campo “id” como chave primária e auto increment. Nas linhas 9 e 11, indicam que os atributos, “nome” e “código”, também serão campos da tabela. E na linha 13 é para informar que este campo será uma chave estrangeira, ou seja, que se relacionará com outra tabela (entidade). Assim como no SQLite, para facilitar a inserção de registros no banco dados, é possível utilizar construtores, conforme as linhas 17 e 20. As linhas abaixo dos construtores, são apenas os métodos de acesso, *get* e *set*.

Apesar do exemplo (Figura 2) retratar poucos tipos de dados, também é possível trabalhar com outros tipos primitivos (boolean, byte, short, int, long, float, double), e também suporta String, Date e byte[]. (WATSON, 2016).

A figura 3, mostra o exemplo da *notation* `@ForeignKeyCollectionField`.



```

13  @DatabaseTable(tableName = "estudante")
14  public class Estudante {
15      @DatabaseField(generatedId = true) //id auto increment
16      private int id;
17      @DatabaseField
18      private String nome;
19      @ForeignKeyCollectionField
20      private Collection<Disciplina> disciplinas;
21
22      public Estudante() {
23      }
24
25      public Estudante(int id, String nome) {
26          this.id = id;
27          this.nome = nome;
28      }

```

Figura 21 - Usando Notations para persistir a classe Estudante

Nas linhas 19 e 20, é finalizado o relacionamento entre as entidades “disciplina” e “estudante”. Desta forma, indicando para o ORMLite, que um estudante poderá estar em várias disciplinas, e em uma disciplina poderá ter apenas um estudante. De fato, na realidade não é isso o que acontece, para este caso teria que ser uma relação muitos para muitos, onde um estudante poderá estar em várias

disciplinas, e uma disciplina poderá ter vários estudantes. Mas como o objetivo é mostrar o funcionamento do *framework*, isso não vem ao caso. Na figura 4, temos o modelo relacional que foi criado com essas duas classes (Disciplina e Estudante). Mas reforçando, o Modelo de Entidades e Relacionamentos (MER) ilustrado abaixo, foi criado pelo próprio *framework* em estudo.

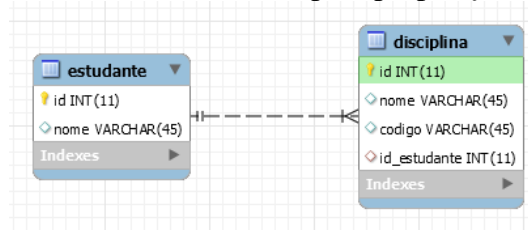


Figura 22 - MER

2.3 Conexão com o bando de dados

No ANEXO 1 é ilustrado a classe que criará o banco de dados (BD), e também responsável por conectar a aplicação ao mesmo.

Como o *framework* trabalha com o padrão Data Access Object (DAO)¹. Após ter criado a classe da conexão, é preciso criar as classes DAO de cada entidade envolvida no projeto, conforme as figuras 5 e 6.

```

12 public class DisciplinaDAO extends BaseDaoImpl<Disciplina, Integer> {
13     public DisciplinaDAO(ConnectionSource cs) throws SQLException {
14         super(Disciplina.class);
15         setConnectionSource(cs);
16         initialize();
17     }
18 }

```

Figura 23 - Classe DisciplinaDAO

```

12 public class EstudanteDAO extends BaseDaoImpl<Estudante, Integer> {
13     public EstudanteDAO(ConnectionSource cs) throws SQLException {
14         super(Estudante.class);
15         setConnectionSource(cs);
16         initialize();
17     }
18 }

```

Figura 24 - Classe EstudanteDAO

2.4 Operações com o banco de dados

Na explicação das operações no BD, será utilizado uma Activity², contendo as variáveis apresentadas na figura abaixo.

```

20 private DatabaseHelper databaseHelper;
21 private EstudanteDAO estudanteDAO;
22 private DisciplinaDAO disciplinaDAO;
23 private List<Estudante> estudantes;
24 private Estudante estudante;
25 private int primeiroID = 0;
26 private static final String CATEGORIA = "Script";

```

Figura 25 - Declaração das variáveis

Dando destaque apenas as linhas 20, 21 e 24, que conforme pode ser visto na Figura 7, instanciam as classes “DatabaseHelper”, “EstudanteDAO”, “DisciplinaDAO” e “Estudante”, respectivamente. E na linha 23 temos um *ArrayList* que guardará uma lista de estudantes.

¹ DAO é um padrão de projeto que abstrai e encapsula os mecanismos de acesso a dados, escondendo os detalhes da execução da origem dos dados. (MORENO; SILANS, 2009).

² Activity é um módulo único e independente que está relacionada com uma tela de interface de usuário e suas funcionalidades correspondentes. (Android Developers, 2017).

Para todas as operações no BD, como inserções, consultas, alterações, logicamente será preciso ter uma conexão, que conforme ilustrado na figura 8, pode ser feita da seguinte forma.

```

33 //CONEXÃO
34 try {
35     databaseHelper = new DatabaseHelper(MainActivity.this);
36     estudanteDAO = new EstudanteDAO(databaseHelper.getConnectionSource());
37     disciplinaDAO = new DisciplinaDAO(databaseHelper.getConnectionSource());
38 } catch (SQLException e) {
39     e.printStackTrace();
40 }

```

Figura 26 – Criando os objetos

Na linha 35 é feita a conexão com BD, na linha 36 pega as informações da tabela estudante e passa para o objeto “estudanteDAO”, e na linha 37 pega as informações da tabela disciplina e passa para o objeto “disciplinaDAO”.

Daqui em diante será ilustrado algumas operações do banco de dados orientado a objetos, ORMLite.

```

42 estudantes = new ArrayList<Estudante>();
43
44 estudante = new Estudante();
45 estudante.setNome("Alex");
46 estudante.setDisciplinas(Arrays.asList(new Disciplina("Matemática", "MT"), new Disciplina("Português", "PT")));
47 estudantes.add(estudante);
48
49 estudante = new Estudante();
50 estudante.setNome("Tiago");
51 estudante.setDisciplinas(Arrays.asList(new Disciplina("Geografia", "GE"), new Disciplina("Artes", "AR")));
52 estudantes.add(estudante);
53
54 //INSERINDO TODOS OS REGISTROS QUE ESTÃO DENTRO DO ArrayList
55 for (Estudante estud : estudantes){
56     int result = estudanteDAO.create(estud);
57     if(result == 1){
58         for (Disciplina disc : estud.getDisciplinas()){
59             disc.setEstudante(estud);
60             disciplinaDAO.create(disc);
61         }
62         primeiroID = primeiroID == 0 ? estud.getId() : primeiroID;
63     }
64 }

```

Figura 27 - Inserindo todos os registros da lista no BD

Das linhas 44 até 47, insere no *ArrayList* o estudante “Alex”, que cursa as disciplinas “Matemática” e “Português”, das linhas 49 até 52, insere no *ArrayList* o estudante “Tiago”, que cursa as disciplinas “Geografia” e “Artes”. Em seguida é percorrido todos os alunos e suas respectivas disciplinas salvas no *ArrayList*, e inseridas no banco dados. Quando é inserido um registro em forma de objeto, é preciso que o objeto DAO chame o método create, e passe o estudante ou disciplina que será inserido(a), conforme nas linhas 56 e 60.

Desta forma é possível perceber que se fosse para inserir os registros um por um, simplesmente poderia usar, “objetoDAO.create(objeto)”, que no exemplo estudado seria, “estudanteDAO.create(estudante)” ou “disciplinaDAO.create(disciplina)”.

```

65 //SELECIONADO OS REGISTROS DO BANCO
66 Log.i(CATEGORIA, "REGISTROS CADASTRADOS");
67 estudantes = estudanteDAO.queryForAll();
68 //Pegando todas as linhas
69 for (Estudante estud : estudantes){
70     Log.i(CATEGORIA, "Nome: "+estud.getNome()+"\nID: "+estud.getId() + "Disciplinas: " + estud.getDisciplinas().size());
71     //Listando as disciplinas do aluno
72     for (Disciplina disc : estud.getDisciplinas()){
73         Log.i(CATEGORIA, "Disciplina: "+disc.getNome()+"\nID: "+disc.getId() + " Código: " + disc.getCodigo());
74     }
75 }

```

Figura 28 - Selecionado os registros

Na linha 67, podemos perceber o uso do método *queryForAll*, responsável por pegar todos os registros de uma determinada entidade. Outra forma de consultar dados, seria através do método *queryRaw*, que permite consultar os registros através de Scripts SQL, conforme pode ser visto na figura 11.

```

79 //LISTANDO TODOS OS REGISTROS, ATRAVÉS DO MÉTODO RAW
80 Log.i(CATEGORIA, "CONSULTANDO REGISTROS ATRAVÉS DO MÉTODO RAW");
81 GenericRawResults<Estudante> raw = estudanteDAO.queryRaw("SELECT id, nome FROM estudante WHERE nome LIKE \"Alex*\"",
82     new RawRowMapper<Estudante>() {
83         //Construtor que recebe como parâmetro o array de colunas, e o array dos valores, respectivamente
84         //Retorna todos os registros
85         @Override
86         public Estudante mapRow(String[] arrayColunas, String[] arrayValores) throws SQLException {
87             return new Estudante(Integer.parseInt(arrayValores[0]), arrayValores[1]);
88         }
89     });
90 //For para percorrer todos os registros encontrados pelo mapRow
91 for (Estudante estud : raw){
92     Log.i(CATEGORIA, "Nome: "+estud.getNome()+"\nID: "+estud.getId());
93 }

```

Figura 29 - Consultas com o método *queryRaw*

Na linha 81 pode-se notar o Script SQL que foi passado através do método *queryRaw*, por sua vez este método exige a criação de um método construtor (linha 86), que recebe como parâmetro um *array* de campos (atributos), e um *array* dos valores de seus respectivos campos. Para percorrer todos os registros encontrados, pode ser utilizado iterações, de acordo com a linha 91.

```

95 //CONSULTANDO REGISTROS PELO ID
96 Log.i(CATEGORIA, "CONSULTANDO REGISTROS PELO ID");
97 estudante = estudanteDAO.queryForId(1);
98 //Deletando estudante encontrado através do ID
99 estudanteDAO.delete(estudante);

```

Figura 30 - Excluindo registro com ID = "1"

Primeiramente é preciso saber qual registro será excluído, e isso acontece na linha 97, onde é usado o método *queryForId*, que permite consultar registros através do ID, neste caso é o ID "1". Depois de passar o resultado para o objeto "estudante", é possível excluir um registro específico, e é o que acontece na linha 99.

```

99 //CONSULTANDO REGISTROS PELO ID
100 Log.i(CATEGORIA, "CONSULTANDO REGISTROS PELO ID");
101 estudante = estudanteDAO.queryForId(2);
102 //Alterando o nome do estudante encontrado através do ID
103 estudante.setNome("Tiago José Coelho");
104 estudanteDAO.update(estudante);

```

Figura 31 - Alterando o nome do registro com ID = 2

Semelhante ao caso anterior, primeiramente é preciso saber qual registro será alterado, e isso acontece na linha 101, onde é usado o método *queryForId*, que permite consultar registros através do ID, neste caso é o ID "2". Depois de passar o resultado para o objeto "estudante", é possível alterar um registro específico, e é o que acontece na linha 104.

Aqui se encerra a apresentação das funções do ORMLite, pelo fato de ter mostrado as principais funcionalidades, e também por ser um *framework* muito amplo.

3. Conclusão

O principal objetivo deste trabalho era mostrar como funciona o banco de dados orientado a objetos, ORMLite, sendo que este foi atendido, pois o artigo ilustra as principais operações em um banco de dados, inserir, listar, alterar e excluir.

Interessante deste framework, é que tem a possibilidade de escolher de qual forma que será feita a operação, seja exclusivamente através de objetos, ou por Script SQL. Adequado para quem não está muito habituado com este tipo de tecnologia, e prefere usar os velhos e famosos Scripts. Mas não apenas por isso, imagine uma consulta com várias relações, fazer isso somente com objetos, seria muito trabalhoso.

Outro ponto relevante, é que se precisar criar ou alterar uma entidade no banco de dados, basta criar ou alterar uma classe, e deixar que o ORMLite faça o resto. Facilitando e agilizando o trabalho do desenvolvedor.

Neste artigo não foi apresentado todas as suas funcionalidades, pois possui várias, e muitas delas não seriam colocadas em prática. Tendo em mente que as que foram apresentadas, são as que normalmente uma aplicação que se conecta a um banco de dados precisa apresentar.

Referências

Elmasri, R. e Navathe S. B. **Sistemas de Banco de Dados**. 6 a edição. Editora Addison-Wesley, (2011).

Mansueli, V. A. P. (2016) **Bancos de Dados Orientados a Objetos - SQL Magazine 78**.

Disponível em: <<http://www.devmedia.com.br/bancos-de-dados-orientados-a-objetos-sql-magazine-78/17717>>. Acesso em: 23 maio 2017.

SQLite (2017). Site oficial do SQLite. Disponível em: <<http://www.sqlite.org/>>. Acesso em: 23 maio 2017.

Moreno, Bruno Neiva; Silans, Alain Passerat De. **Implementação de um sistema de informação para tratamento dos dados do BEER**. João Pessoa-PB, 2009.

Neto, José Dijon de O.; Maia, Iverton P. L.; Lemos, Emilio C. L.; Castro, Angélica Félix de.

Análise comparativa de desempenho de aplicação Android com persistência em Banco de Dados Relacional e Banco de Dados Orientado a Objetos. Mossoró – RN, 2017.

Watson, Gray. (2016). **ORMLite Package**. Disponível em: <<http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite.html>>. Acesso em: 31 maio 2017.

Anexos

```

18 public class DatabaseHelper extends OrmliteSqliteOpenHelper{
19     private static final String databaseName = "banco_de_dados_escola.db";
20     private static final int databaseVersion = 13;
21
22     public DatabaseHelper(Context context) { super(context, databaseName, null, databaseVersion); }
23
24
25
26     @Override
27     public void onCreate(SQLiteDatabase sqLiteDatabase, ConnectionSource connectionSource) {
28         try {
29             TableUtils.createTable(connectionSource, Estudante.class);
30             TableUtils.createTable(connectionSource, Disciplina.class);
31         } catch (SQLException e) {
32             e.printStackTrace();
33         }
34     }
35
36     @Override
37     public void onUpgrade(SQLiteDatabase sqLiteDatabase, ConnectionSource connectionSource, int oldVersion, int newVersion) {
38         try {
39             TableUtils.dropTable(connectionSource, Estudante.class, true); // o true vai fazer com que não apareça os erros
40             TableUtils.dropTable(connectionSource, Disciplina.class, true);
41             onCreate(sqLiteDatabase, connectionSource);
42         } catch (SQLException e) {
43             e.printStackTrace();
44         }
45     }
46
47     @Override
48     public void close() { super.close(); }
49
50
51 }

```

Anexo 1

Sistema de Irrigação Automatizado com Interface de Controle e Monitoramento Online

Alex Manoel Coelho¹, Wesley Bezerra²

¹Aluno 7ª fase do Curso de Ciência da Computação do Instituto Federal Catarinense –
Campus Rio do Sul

²Professor de Ensino Superior do Instituto Federal Catarinense – Campus Rio do Sul
alexma_coelho@hotmail.com.br, wesley.bezerra@ifc.edu.br

Abstract. Nowadays we find many solutions of irrigation for the agricultural sector, but, nevertheless, many of these technologies present a very high cost, or even because they are solutions for large plantations, making them unviable for the use of the small producers. Irrigation methods that apply to these small farmers, despite their efficiency and cost-effectiveness, do not present current technological solutions, still maintaining primitive characteristics in their application, for this problem, we seek to present a low-cost solution with resource use Technologies.

Key-words: Arduino; Automation; Agriculture.

Resumo. Atualmente encontramos muitas soluções de irrigação para o setor agrícola, mas, no entanto, muitas dessas tecnologias apresentam um custo muito elevado, ou até por serem soluções para grandes plantações, as tornando inviável para o uso dos pequenos produtores. Os métodos de irrigação que se aplicam a esses pequenos agricultores, apesar de sua eficiência e custo-benefício, não apresentam soluções tecnológicas atuais, ainda mantendo características primitivas em sua aplicação, para tal problema, buscamos apresentar uma solução de baixo custo e com uso de recursos tecnológicos atuais.

Palavras-chave: Arduino; Automação; Agricultura.

4. Introdução

Será tratado nesse trabalho de um sistema de irrigação automatizado, controlado por uma interface *web*, que tem como finalidade criar um sistema de controle da irrigação acessível e simples para os pequenos agricultores.

Tendo em vista o fato que a região do Alto Vale do Itajaí é extremamente agrícola e é dominada pela agricultura familiar, foi vista à necessidade de criar um sistema para a irrigação dessas propriedades, que seja acessível a essas famílias de agricultores, e ainda possibilite mais agilidade no gerenciamento e ativação da irrigação, não importando o local onde o operador da irrigação se encontre, sendo somente necessária uma conexão com a internet.

O sistema tem como objetivo, verificar as condições ambientais através de sensores e a partir da combinação dos dados climáticos obtidos, definir o melhor

momento para iniciar a irrigação, além de armazenar todas as configurações e histórico de monitoramento e irrigação em seu banco de dados.

Ele permitirá uma fácil adaptação a irrigação de várias culturas distintas, a partir da inserção dos dados climáticos necessários para a irrigação.

4.1 Soluções Existente

Podemos encontrar várias soluções já existentes em sistemas de irrigação de baixo custo, no entanto algo que encontramos em comum em todas elas, é o baixo nível tecnológico das mesmas.

Hoje programas governamentais voltados a apoio à agricultura, fornecem cartilhas com técnicas de irrigação de baixo custo. Desses documentos o mais facilmente encontrado é uma cartilha sobre um Sistema de Manejo de Irrigação para a Agricultura Familiar, feito pela Embrapa.

Notamos que a cartilha da Embrapa, tem foco para a irrigação em localidades com recursos hídricos escassos e para agricultura de subsistência. Outro ponto que notamos é que as técnicas apresentadas se resumem somente ao maquinário utilizado e as técnicas são principalmente manuais, sem uso de tecnologia computacional, vemos também um foco em utilização de matérias reaproveitados, cremos que esse sistema possa ser complementado por nossa proposta apresentada. (COELHO et al., 2014)

4.2 Proposta

Para tentar solucionar essa problemática nos sistemas de irrigação de baixo custo, utilizamos nossos conhecimentos adquiridos dentro de nossa graduação para criar uma proposta.

Utilizamos conhecimentos adquiridos na matéria de Sistemas Digitais em sistemas de microcontroladores, utilizamos o conhecimento em programação orientada a objetos obtidas na matéria de Linguagem de Programação I, conhecimento em banco de dados adquiridos nas disciplinas de Banco de Dados e por fim os conhecimentos em programação WEB obtidos na matéria de Linguagem de Programação II.

Nossa proposta se trata de um sistema de irrigação, utilizando um conjunto de sensores controlados por um Arduino, esse conjunto de sensores é posto na área onde se deseja realizar o monitoramento, esses conjuntos de sensores são cadastrados em um sistema WEB, onde o usuário poderá informar a sua localização, bem como as configurações com os requisitos para iniciar uma irrigação, além de o usuário poder acompanhar a situação de sua irrigação não somente localmente, mas também de forma remota. O sistema ainda armazena todo o histórico de irrigação e de configurações em um banco de dados.

5. Projeto

5.1 Objetivo Geral

O principal objetivo é apresentar uma proposta de sistema de irrigação de baixo custo, para pequenos agricultores com recursos escassos.

Temos também como objetivo poder demonstrar de forma prática os conhecimentos apresentados na disciplina de Linguagem de Programação II.

5.2 Objetivos Especificos

Traçamos alguns objetivos parciais, onde sua conclusão acaba por formar o sistema completo.

O primeiro objetivo parcial é a montagem e configuração dos sensores com o Arduino, tarefa essa simples, visto que esse objetivo já foi atingido na disciplina de Sistemas Digitais.

Outro objetivo foi a criação de uma página web visualmente agradável e com boa usabilidade, para a criação de tal foi utilizado o framework Bootstrap, que possibilita a criação de uma página bonita, funcional e ainda responsiva.

Tivemos o objetivo de criar um Back-End robusto o suficiente para gerenciar o sistema online, bem como gerenciar o banco de dados, com adição, remoção e alteração das informações armazenadas no mesmo. Para tal foi utilizado o sistema JSP (Java Server Pages), um sistema para o Back-End baseado na linguagem de programação Java

Outro ponto foi a utilização de um banco de dados para o armazenamento dos dados de irrigação bem como as configurações da mesma, nesse ponto foi optado por utilizar o sistema de banco de dados MySQL, sistema no qual aprendemos a utilizar nas disciplinas de Banco de Dados e utilizamos no conteúdo didático em Linguagem de Programação II.

O objetivo mais importante é a junção de todos os objetivos anteriores, com a junção de tudo temos nosso sistema de irrigação completo para o uso.

6. Metodologia

Para realização deste projeto foi utilizado um conjunto de várias ferramentas e equipamentos.

Para a parte de do sistema de monitoramento, foram utilizados um conjunto de sensores gerenciados por um controlador, que com a combinação dos dados informados pelos sensores, ativa ou desativa um relé que controla o acionamento do maquinário responsável pela irrigação:

Para a obtenção de dados foram utilizados um conjunto de sensores que segue abaixo:

6.1 Sensor DHT11:

Usado para realizar a leitura da Temperatura e da Umidade do ar. É um sensor de umidade relativa e temperatura, com saída digital calibrada. Possui uma exclusiva tecnologia para medir a umidade, garantindo a confiabilidade e estabilidade. Possui internamente um microcontrolador de 8 Bits para tratar o sinal. Possui tamanho compacto, baixo consumo, encapsulamento simples com apenas quatro terminais, podendo transmitir o sinal por até 20 metros. Todas essas características permitem utilizar esse sensor nas mais diversas aplicações. (CARVALHO, 2013).

6.2 Sensor Moisture Sample:

Usado para realizar a leitura da humidade do solo. Este sensor pode ler a quantidade de umidade presente no solo ao seu redor. Este sensor utiliza as duas sondas para passar corrente através do solo, e, em seguida, a resistência faz uma leitura para obter o nível de umidade. O fator principal é a água, pois conduz eletricidade mais facilmente (menor resistência), e sem água e com o solo seco a condutividade de energia é menor (maior resistência). (CARVALHO, 2013).

6.3 Relay

Tem a capacidade de controlar diferentes tensões e potências sem o risco de danificar o micro-controlador. (CARVALHO, 2013)

6.4 Arduino Uno R3

É uma placa de microcontrolador, ele tem 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, um cristal oscilador de 16MHz, uma conexão USB, uma entrada de alimentação uma conexão ICSP e um botão de reset. (CARVALHO, 2013).

Para a criação da interface *WEB* usamos basicamente HTML, CSS e JavaScript, para agilizar a confecção e facilitar a criação do código da interface utilizamos o *Framework* Bootstrap.

O “Framework” Bootstrap, é uma biblioteca CSS e HTML que fornece *templates* para a tipografia, botões, navegação e componentes de navegação em geral, onde o usuário basta chamar a classe em uma determinada *tag* HTML para o visual ser aplicado na página. (TWITTER, 2016)

Para o *Back-End* acabamos por utilizar o sistema JSP, que necessita de vários componentes para poder ser utilizado, no caso desse projeto os componentes foram os seguintes (SESHADRI, 1999):

- a) JDK (*Java Development Kit*);
- b) Netbeans IDE;
- c) Apache Tomcat Server.

O JDK é o conjunto de componentes para o desenvolvimento na linguagem de programação Java. Ele é composto pela JVM (*Java Virtual Machine*), que é uma máquina virtual, que compila o código Java para um código que pode ser lido por qualquer máquina, também é composto ainda por um conjunto de bibliotecas com funcionalidades que facilitam a programação em JAVA. (ORACLE, 2013).

O Netbeans IDE é um ambiente de desenvolvimento integrado que possibilita a o desenvolvimento em uma série de linguagens, que nesse caso em específico foi utilizado para desenvolvimento em Java, ele oferece as ferramentas necessárias para o desenvolvimento tanto de aplicações *Desktop*, como desenvolvimento *WEB*. Em sua instalação é possível instalar junto todos os componentes para o desenvolvimento das aplicações, nesse caso em específico optamos por instalar o Netbeans com as ferramentas necessárias para o desenvolvimento em JSP, o que inclui em sua instalação o Apache Tomcat. (ORACLE, 2016).

O Apache Tomcat é um servidor WEB Java, que implementa as tecnologias Java Servlet e JavaServerPages. Ele proporciona um ambiente onde os códigos Java puros, podem ser executados para a confecção de sistemas WEB com essa linguagem. (TOMCAT FOUNDATION, 2013).

O último ponto necessário para o sistema é o seu banco de dados, no qual foi utilizado o sistema gerenciador de Banco de Dados MySQL. Se trata de um sistema de banco de dados muito difundido mundialmente, principalmente devido a sua fácil integração com a linguagem de programação PHP, essa sendo a linguagem de programação mais utilizada para criação de sites para a internet. É um sistema de banco de dados muito funcional, principalmente por sua portabilidade, pois roda em diversos sistemas operacionais e por sua compatibilidade, por suportar diversas aplicações que rodam em uma grande diversidade de linguagens de programação e numa diversidade de plataformas. (ORACLE, 2016).

6.5 Desenvolvimento

Para a aplicação do projeto foram criadas uma série de classes para a manipulação das informações do sistema, o diagrama com essas classes pode ser visto abaixo.

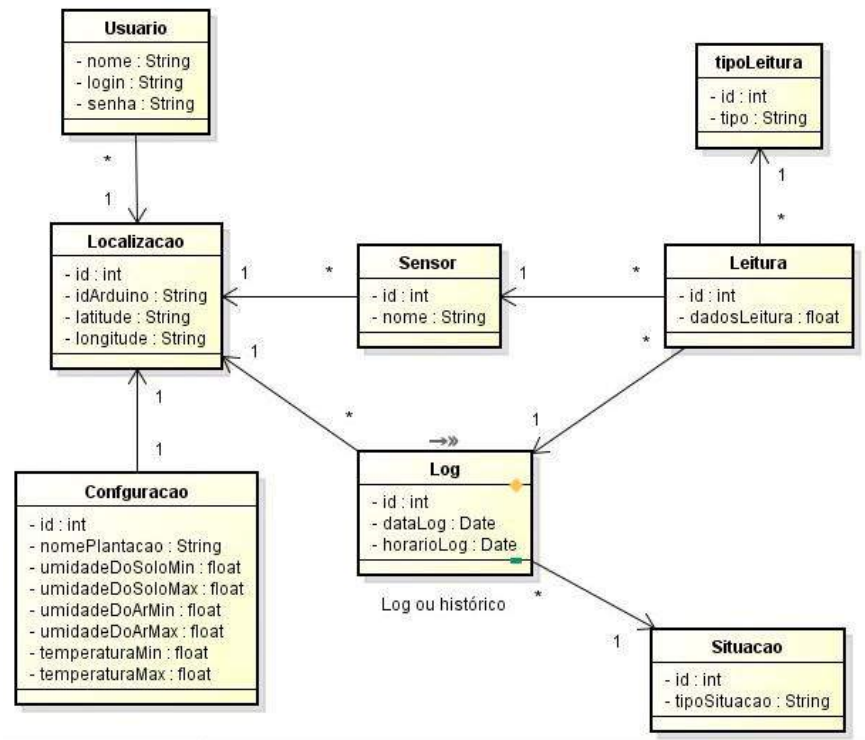


Figura 32 - Diagrama de Classes

Para o banco de dados do sistema podemos ver as tabelas e utilizadas e suas ligações demonstradas na figura abaixo (Figura 2).

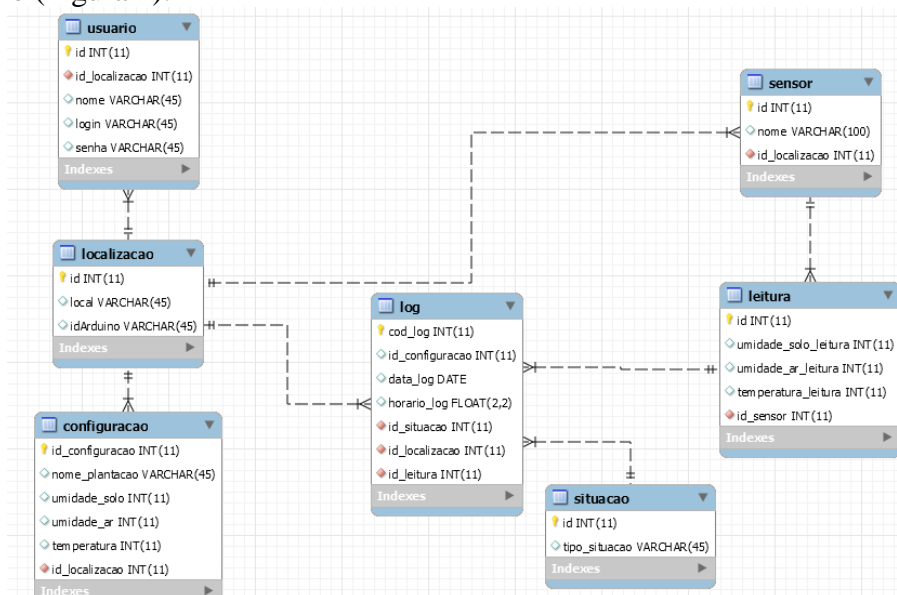


Figura 33 - Modelo de Entidades e Relacionamentos

7. Conclusão

A proposta inicial do projeto, foi a confecção de um sistema de irrigação automatizado de baixo custo, utilizando uma controladora Arduino, juntamente com sensores, onde o gerenciamento do sistema seria realizado por uma interface WEB, para dessa forma facilitar a atividade dos pequenos agricultores.

De um modo geral, esse trabalho acabou por envolver diversos tópicos visto em todo o curso de Bacharelado em Ciências da Computação, sobre Sistemas Digitais, Banco de Dados e Programação, ainda estendendo os assuntos para a área da agricultura.

Para a conclusão do projeto, houve um grande aprendizado sobre a confecção de páginas para a internet, do funcionamento dos sistemas WEB, além de demonstrar a versatilidade da linguagem de programação Java, podemos ver ainda demonstrar o uso de banco de dados em uma situação real. A criação de um sistema de sensores controlador por Arduino, além da confecção de todo o sistema WEB, bem como a utilização de um banco de dados para o armazenamento das informações da irrigação, foram etapas, concluídas com êxito, sem nenhum grande problema que impedisse a sua implementação.

Visto isso, podemos concluir que, é possível criar um sistema de irrigação moderno e com baixo custo em sua confecção e aplicação, graças pelo grande range de medições que esses sensores simples possuem, é totalmente possível os adaptar para diversas culturas, em diversas condições, portanto o trabalho foi concluído com êxito.

Referências

COELHO, Eugênio Ferreira et al. **Sistemas e Manejo de Irrigação de Baixo Custo para Agricultura Familiar**. 2014. Disponível em:

<<http://ainfo.cnptia.embrapa.br/digital/bitstream/item/133043/1/Cartilha-Manejo-Irigacao-03-09-2015.pdf>>. Acesso em: 02 dez. 2016.

CARVALHO, Renon Steinbach. **Sistema de Irrigação Automatizado**. 2013. 12 f. Ifc - Rio do Sul, Rio do Sul, 2013.

TWITTER (Estados Unidos) (Org.). **Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web**. 2016. Disponível em: <<http://getbootstrap.com/>>. Acesso em: 02 dez. 2016.

SESHADRI, Govind. **Understanding JavaServer Pages**. 1999. Disponível em: <<http://www.javaworld.com/article/2076557/java-web-development/understanding-javascript-pages-model-2-architecture.html>>. Acesso em: 02 dez. 2016.

ORACLE (Estados Unidos). **Java SE 7 Features and Enhancements**. 2013. Disponível em: <<http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html>>. Acesso em: 02 dez. 2016.

ORACLE (Estados Unidos). **NetBeans Release Roadmap**. 2016. Disponível em: <<https://netbeans.org/community/releases/roadmap.html>>. Acesso em: 02 dez. 2016.

TOMCAT FOUNDATION (Estados Unidos). **Apache Tomcat Configuration Reference**. 2013. Disponível em: <<http://tomcat.apache.org/tomcat-5.5-doc/config/realm.html>>. Acesso em: 02 dez. 2016.

COMPARATIVO DO USO DE LINGUAGENS DE PROGRAMAÇÃO E GERADORES DE CÓDIGO NO DESENVOLVIMENTO DE SISTEMAS

Jean Felipe Diel¹, Luiz Cláudio Dalmolin²

¹ Universidade do Estado de Santa Catarina (UDESC)

Centro de Educação Superior do Alto Vale do Itajaí (CEAVI)

² Universidade do Estado de Santa Catarina (UDESC)

Centro de Educação do Planalto Norte (CEPLAN)

jfdiel@gmail.com, luiz.dalmolin@udesc.br

Abstract. Choose the best platform or technology for use in developing a system requires a number of factors are analyzed. A wrong choice can compromise the delivery and future maintenance of the final product. No documents or templates to follow and help when choosing which adopt technology to build a system. Thus, this article aims to highlight advantages and disadvantages as well as positives and negatives of using a code generator or a programming language in the software development stage. For comparison, PHP and Java programming languages were adopted, those used in the Upper Itajaí Valley region. Companies in the region chose the used generators, GeneXus and Scriptcase because they generate their code in Java and PHP, respectively, and for use.

Keywords: Programming language, code generators, PHP, Java, Scriptcase, GeneXus.

Resumo. Escolher a melhor plataforma ou tecnologia para se utilizar no desenvolvimento de um sistema requer que uma série de fatores sejam analisados. Uma escolha errada pode comprometer a entrega e futuras manutenções no produto final. Não há documentos ou modelos para seguir e ajudar no momento de escolher qual tecnologia adotar para construir um sistema. Logo, este artigo tem como objetivo destacar vantagens e desvantagens bem como pontos positivos e negativos de se utilizar um gerador de código ou uma linguagem de programação na etapa de desenvolvimento de software. Para o comparativo, foram adotados as linguagens de programação PHP e Java, estas utilizadas na região do Alto Vale do Itajaí. Os geradores utilizados, GeneXus e Scriptcase, foram escolhidos por gerarem seus códigos em Java e PHP respectivamente e por serem utilizados por empresas da região.

Palavras-chave: Linguagem de programação, geradores de código, PHP, Java, Scriptcase, GeneXus.

1. Introdução

Com o passar dos anos é possível observar que o mercado da informática vem se tornando cada vez mais competitivo e exigente. Os usuários e clientes buscam sistemas

mais dinâmicos e flexíveis, mas que também sejam desenvolvidos em curto prazo e principalmente, que necessitem de pouco investimento.

Na hora de desenvolver uma aplicação, vários fatores devem ser levados em consideração pela equipe. Ao levantar os requisitos, tanto cliente, quanto membros do projeto, imaginam como deverá ficar o sistema ao final do processo de construção. Porém, se o desenvolvimento falhar ou construir algo distinto do planejado, pode comprometer todo o projeto e relação junto ao cliente (O'KEEFFE, 2012).

Escolher a ferramenta e linguagem a ser adotada, se esta ainda não estiver definida, deve ser considerada como uma etapa importante dentro do processo de desenvolvimento. Errar nessa escolha, pode acarretar não só em problemas durante a construção da aplicação mas em futuras melhorias e manutenções. Todos os envolvidos na etapa de desenvolvimento tem de conhecer ou ter o mínimo de conhecimento da ferramenta escolhida (VAREJÃO, 2004).

Embora leva-se quase sempre em consideração o conhecimento dos membros do projeto, é preciso que a equipe procure em outros ambientes possíveis problemas que possam vir a ocorrer, tais como: falta de mão de obra, carência de suporte da linguagem ou da ferramenta e custos elevados.

Este trabalho tem por objetivo apontar aspectos positivos e negativos, bem como vantagens e desvantagens entre o uso de ferramentas geradoras de código fonte e linguagens de programação no ambiente de desenvolvimento. Foram adotadas as linguagens de programação Java e PHP, levando-se em consideração que ambas são ensinadas e utilizadas na região do Alto Vale do Itajaí, onde encontra-se o campus da Udesc. Para as ferramentas geradoras de código, optou-se para comparação, as ferramentas GeneXus e Scriptcase, também utilizadas na região e que geram seus códigos em Java e PHP respectivamente.

2. Linguagens de Programação

Uma linguagem de programação pode ser definida segundo Tucker (2007), como uma forma de comunicação de ideias entre humanos e computadores, porém com um domínio de expressão mais reduzido do que as linguagens naturais. Assim, a linguagem de programação traduz as ideias humanas em comandos compreendidos pelos computadores.

Muitas linguagens de programação foram criadas ao longo dos anos. Para Aguillar (2011), as linguagens de programação servem de base para a escrita de algoritmos, cujo objetivo é solucionar problemas por meios computacionais. Essa escrita por meio de uma linguagem, quase sempre baseada em palavras no idioma inglês, facilita a comunicação humano-computador.

3. Geradores de Código Fonte

Um gerador de código pode ser definido como uma ferramenta que ao receber uma entrada de dados de forma estruturada, retorna como saída, o código fonte em uma linguagem definida, dispensando trabalho manual de um programador. Um gerador de código pode ser traduzido como um modelo a ser seguido pelos programadores, que uma vez escrito, gerará o código fonte sempre da mesma forma (HUNT; THOMAS, 1999).

Para Martins (2007), gerar o código de uma aplicação manualmente, pode acarretar em diversos problemas para a equipe de desenvolvimento. Quando escrita sem um gerador, o código

pode ter sua qualidade aumentada pelo programador a cada manutenção, porém, com o uso de um gerador, o código é gerado de forma automática e a qualidade será consistente.

Os maiores benefícios do uso de geradores de código são a qualidade: todo o código gerado seguirá um *template* bem escrito e testado; consistência: há um padrão entre as classes, já que se segue o mesmo *template* e estes por sua vez, recebem todas as decisões de arquitetura; único ponto de conhecimento: para mudanças na aplicação, basta alterar o *template* ou elementos de entrada do gerador; e por fim, foco no *design*: todo código repetido é gerado de forma automática, o programador tem de focar apenas no conhecimento e análise (MARTINS, 2007).

3.1 GeneXus

Criada pela empresa uruguaia Artech, o GeneXus, lançado no ano de 1988, é uma ferramenta direcionada à criação de aplicações Web. Desenvolvida para o sistema operacional Windows e permite a geração de código para as linguagens C#, Java e Ruby (GENEXUS, 2016).

Segundo Genexus (2016), a partir da modelagem do sistema desejado, o GeneXus cria automaticamente o banco de dados, o código fonte do aplicativo, a interface do usuário para o cliente e os serviços necessários para o servidor. A ferramenta permite ainda gerar e utilizar *Web Services*, gerar documentos, requisições HTTP e funções para *e-mail* (SMTP, IMAP, POP3, entre outros) sem exigir muito conhecimento e esforço por parte do desenvolvedor.

A figura 1 ilustra a criação de um relatório utilizando GeneXus. É possível observar que a ferramenta disponibiliza caixas de ferramentas e menus que auxiliam ao programador na criação de aplicações na forma visual sem que precise ter contato com o código fonte. A ferramenta abstrai toda a comunicação com o banco de dados, propriedades e sintaxe gerada, requisitando apenas ao usuário, entradas visuais.

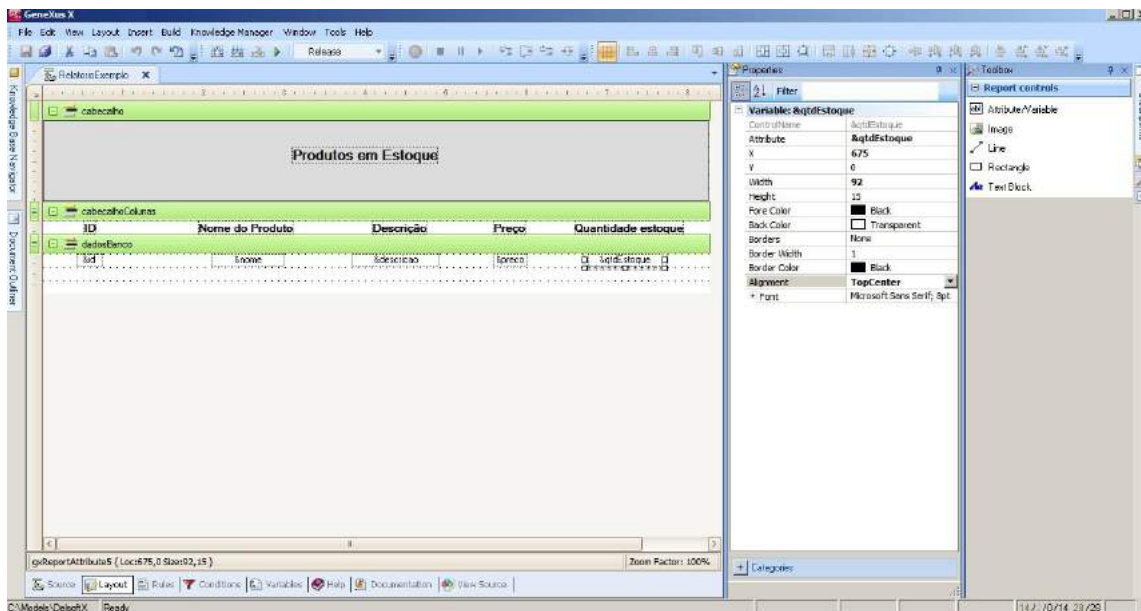


Figura 1 – Criação de um relatório na ferramenta GeneXus.

3.2 Scriptcase

O Scriptcase é um gerador de código PHP, criado em 2000, pela empresa NetMake. Atualmente na versão 8.1, é uma ferramenta multiplataforma para a criação de aplicações Web, que se ajustam a qualquer dispositivo móvel (SCRIPTCASE, 2016).

Segundo Scriptcase (2016), o gerador Scriptcase, pode ter múltiplas conexões com diferentes bancos de dados disponíveis no mercado. Possibilita a importação e exportação de documentos, disponibiliza ferramentas e gráficos para análise de dados, temas para as aplicações, filtros dinâmicos e outras inúmeras funcionalidades.

Na figura 2, pode-se visualizar uma das telas de execução do Scriptcase. Nesta tela de exemplo o usuário tem a opção de escolher o banco de dados e tem como obrigação, informar os parâmetros de conexão. Todos os comandos de conexão e iteração da aplicação do usuário, será posteriormente gerada pelo Scriptcase.

Assim como na tela da figura 2, o Scriptcase procura utilizar em todas as *interfaces*, menus e barras de ferramentas com imagens e textos, que permitam a qualquer usuário a utilização desta ferramenta, mesmo que este usuário tenha pouco conhecimento em programação de sistemas.

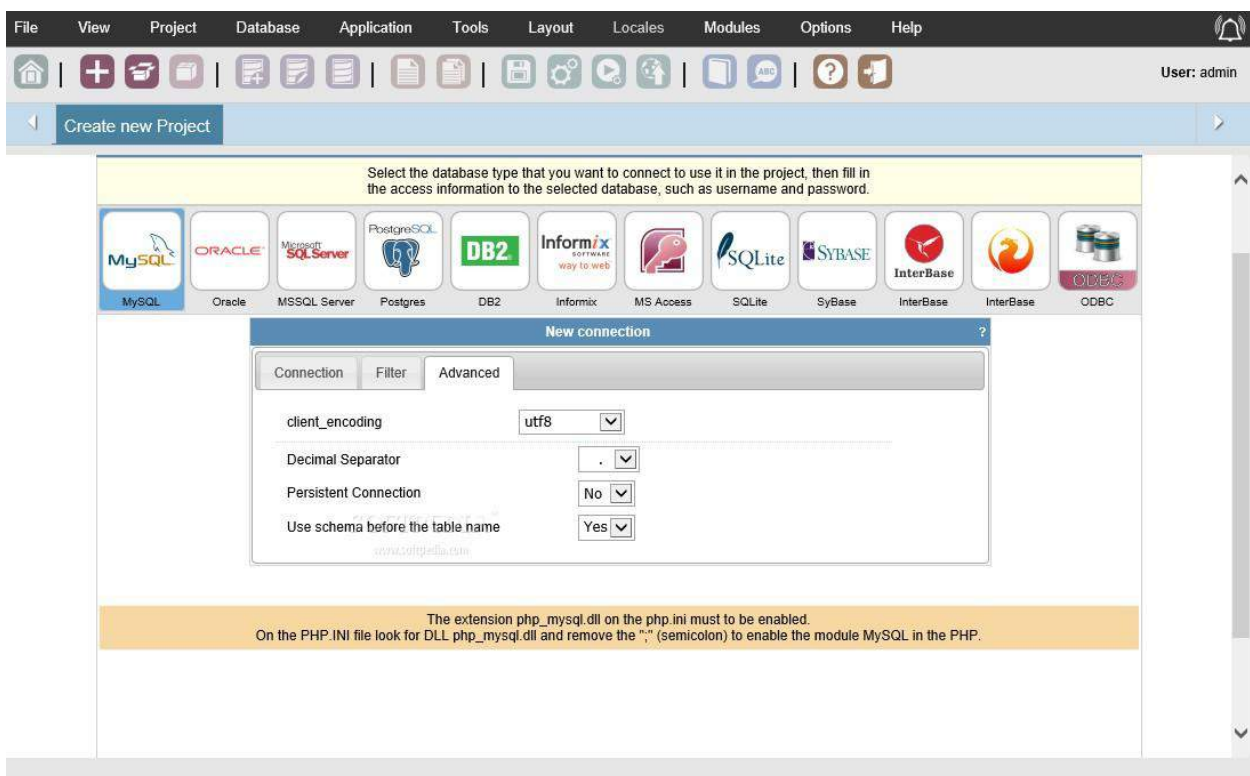


Figura 2 – Tela execução do Scriptcase.

4. Metodologia de comparação

Ao iniciar um novo artefato de software, a equipe de desenvolvimento precisa planejar como conduzirá seu projeto e como o fará. Esse planejamento ocorre principalmente após a etapa de levantamento das regras de negócio e os requisitos funcionais e não funcionais. É com base neles

que serão definidos a linguagem a adotar, o editor ou ambiente de desenvolvimento integrado a utilizar, os recursos e as funcionalidades (GERHARDT, 2003).

O planejamento para escolher uma nova linguagem ou ambiente, pode ocorrer por diversas razões. De acordo com Rezende (2005), todo sistema sofrerá mudanças, sejam elas ajustes após sua implantação ou por substanciais melhorias. Porém, Rezende (2015) afirma que a medida que o sistema passa a sofrer diversas manutenções, o mesmo passa a ficar obsoleto, causando um desgaste do produto para com o usuário final e que a melhor alternativa é construir um novo sistema. Outros fatores podem influenciar a troca de tecnologias ou reconstrução, são mudanças na legislação, incompatibilidade com novos sistemas operacionais, diferentes dispositivos e hardware, documentação ou trocas de membros da equipe.

A exigência para alterar o produto faz com que a equipe precise estudar qual tecnologia adotará. Não por acaso, existem inúmeras situações no mercado de informática em que erros nas estimativas levam a erros na condução do projeto, causando a perda de clientes e de dinheiro (O'KEEFFE, 2012). Por outra vez, muitos processos de migração de linguagem ou de versão de tecnologias chegam ao seu final, mas levam um tempo muito além do esperado.

Muitos fatores devem ser levados em consideração na hora de migrar para uma nova versão do software ou alterar a tecnologia usada. Os profissionais envolvidos no projeto, devem buscar informações além das dependências da empresa. Buscas na Web, entrevistas com outros profissionais que passaram por migrações, construção de protótipos, são técnicas que podem auxiliar para evitar possíveis problemas (GERHARDT, 2003).

Quando a necessidade se dá em relação à linguagem de programação ou gerador de código a serem adotados, a experiência da equipe é sempre determinante. Ao surgir uma nova versão da linguagem ou do gerador de código em que a equipe tenha domínio, os membros precisam apenas se atualizar em relação às novas funcionalidades e correções adotadas. Entretanto, ao se adotar uma tecnologia ainda desconhecida, exige-se que todos aprendam a usá-la, o que toma tempo e exige planejamento para não prejudicar o andamento do projeto (GERHARDT, 2003).

Independente do domínio da tecnologia utilizada pela empresa, os colaboradores tendem a mudar ao decorrer do processo de desenvolvimento. Principalmente na área de informática, onde a troca de membros é comum devido à falta de profissionais qualificados no mercado. A dificuldade em encontrar novos colaboradores aumenta quando a tecnologia usada pela empresa não é a adotada em escolas e universidades e, principalmente, quando há custo significativo para treinamentos no seu uso (PAQUET; MOKHOV, 2010).

Diferentemente das linguagens de programação, uma dificuldade encontrada no uso dos geradores de código é a documentação disponível para auxílio de seus usuários. Assim a documentação fica restrita ao da empresa criadora e de alguns sites de usuários com maior conhecimento. Já as linguagens, dispõem de inúmeros sites oficiais ou não, *blogs*, fóruns e milhares de livros.

Em sua maioria, um gerador de código trata-se de um produto vendido por uma empresa. Assim qualquer problema ou dificuldade em seu uso, o usuário pode contar com o apoio da empresa para auxiliá-lo. O mesmo não ocorre em relação às linguagens de programação, onde há a disponibilidade de documentações, sites oficiais das linguagens e relatos de outros programadores com domínio da linguagem, porém, os membros do projeto precisam dominá-la ou pagar consultorias para alguém que domine o conhecimento sobre essas.

Ao se utilizar uma linguagem de programação para o desenvolvimento de um sistema, precisa-se optar por duas opções, implementar todo o código de forma manual ou utilizar um *framework*. Existe para cada linguagem, diversos *frameworks*. Os *frameworks* disponibilizam diversas funcionalidades implementadas, que podem se necessário serem utilizadas, como interação com banco de dados, serviços de e-mail, *web services*, entre outros, porém exigem domínio da linguagem e conhecimento de alguns paradigmas de programação (ALOMARI ET. AL., 2015).

O uso da linguagem de programação seja ela escrita manualmente ou optando por um *framework*, faz com que as classes sejam criadas pelos membros do projeto, ou seja, ao menos sabe-se como foi escrito e como manter a aplicação. No caso dos geradores de código, o código fonte gerado seguirá um *template* e manutenções só poderão ser feitas utilizando-se o próprio gerador. Sendo assim, o desenvolvedor fica refém das limitações da ferramenta, de forma que qualquer situação além do código gerado que venha a ser solicitada e que não possa ser uma entrada no gerador, ficará pendente por parte da equipe.

Na figura 3, pode-se visualizar o trecho de um código de uma aplicação criada através do gerador de código Scriptcase. O exemplo de aplicação, tem como resultado esperado apresentar uma frase na tela. Mesmo se tratando de uma única linha, o exemplo não deixa de ser uma aplicação, sendo assim, o Scriptcase gera toda a estrutura necessária para caso a aplicação seja continuada.

Se o mesmo código fosse escrito com a linguagem PHP, a frase poderia ser escrita utilizando-se menos de cinco linhas e utilizando apenas um arquivo. O trecho exibido na figura 3, é parte de um arquivo e este por sua vez, faz parte do conjunto de dez arquivos gerados para exibição da frase de exemplo.

```
$SESSION['scriptcase']['charset_entities']['ISO-8859-1'] = 'ISO-8859-1';
$SESSION['scriptcase']['charset_entities']['ISO-8859-5'] = 'ISO-8859-5';
$SESSION['scriptcase']['charset_entities']['ISO-8859-15'] = 'ISO-8859-15';
$SESSION['scriptcase']['charset_entities']['WINDOWS-1251'] = 'cp1251';
$SESSION['scriptcase']['charset_entities']['WINDOWS-1252'] = 'cp1252';
$SESSION['scriptcase']['charset_entities']['BIG-5'] = 'BIG5';
$SESSION['scriptcase']['charset_entities']['EUC-CN'] = 'GB2312';
$SESSION['scriptcase']['charset_entities']['GB2312'] = 'GB2312';
$SESSION['scriptcase']['charset_entities']['SJIS'] = 'Shift_JIS';
$SESSION['scriptcase']['charset_entities']['EUC-JP'] = 'EUC-JP';
$SESSION['scriptcase']['charset_entities']['KOI8-R'] = 'KOI8-R';
$SESSION['scriptcase']['trial_version'] = 'N';
$SESSION['sc_session'][$this->sc_page]['decimal_db'] = ".";

$this->nm_cod_apl = "Exemplo";
$this->nm_nome_apl = "";
$this->nm_seguranca = "";
$this->nm_grupo = "exemplo";
$this->nm_grupo_versao = "1";
$this->nm_autor = "admin";
$this->nm_versao_sc = "v8";
$this->nm_tp_lic_sc = "demo";
$this->nm_dt_criacao = "20160531";
$this->nm_hr_criacao = "234303";
$this->nm_autor_alt = "admin";
$this->nm_dt_ult_alt = "20160531";
$this->nm_hr_ult_alt = "234604";
$temp_bug_list = explode(" ", microtime());
list($NM_usec, $NM_sec) = $temp_bug_list;
$this->nm_timestamp = (float) $NM_sec;
$this->nm_app_version = "1.0.0";

$NM_dir_atual = getcwd();
if (empty($NM_dir_atual))
{
    $str_path_sys = (isset($_SERVER['SCRIPT_FILENAME']) ? $_SERVER['SCRIPT_FILENAME'] : $_SERVER['ORIG_PATH_TRANSLATED']);
    $str_path_sys = str_replace("\\", '/', $str_path_sys);
}
else
{

```

Figura 3 – Exemplo de código gerado pelo Scriptcase.

Ao optar por um gerador de código, as empresas e seus colaboradores podem acabar ficando, ao decorrer dos anos, refém destas ferramentas automáticas. Tomando como exemplo o Scriptcase, o desenvolvedor pode criar programas e mantê-los por anos sem ver uma linha de código durante este tempo. Ao receber um desafio em uma linguagem ou projeto que não exija o uso de um gerador, o mesmo pode encontrar dificuldades.

O uso de uma ferramenta para gerar código, em ambos os casos GeneXus e Scriptcase, ao decorrer do desenvolvimento de aplicações mais robustas, faz com que a geração venha a ficar mais lenta, uma vez que ao implementar uma funcionalidade, ambos geram vários arquivos e assim exigem que o usuário tenha um computador com mais recursos.

Para Varejão (2004) e Sebesta (2011), existe certa dificuldade em se estabelecer critérios para avaliação de uma linguagem e geradores de código, pois muitas vezes esses critérios são subjetivos e até mesmo pessoais. Os critérios podem estar em diferentes níveis de granularidade, podendo ser considerados mais ou menos importantes de acordo com a perspectiva de quem os analisa.

Com base nos critérios de Varejão (2004) e Sebesta (2011), foram elaborados aspectos para o comparativo entre o uso de ferramentas geradoras de código (GeneXus e Scriptcase) e linguagens de programação (PHP e Java), comparando as principais características de cada critério e tecnologia na construção de sistemas. Tal comparativo é demonstrado na tabela 1.

Crítérios	Linguagem de Programação (PHP e Java)	Geradores de código (GeneXus e Scriptcase)
Custo	Não possuem custos. Treinamentos e suportes de terceiros tem custos e qualidade variáveis.	Possuem custo de licença. O treinamento e o suporte estão inclusos na aquisição.
Facilidade de aprendizado	Exige treinamento e prática.	Exige prática, mas em grande maioria, são intuitivos e de fácil aprendizagem.
Fácil manutenção do código	Todo o código é escrito pela equipe.	A ferramenta gera o código na linguagem definida, exige alto conhecimento para manutenções no código gerado pela ferramenta.
Integração com banco de dados	Se não utilizar frameworks, necessita ser implementada toda a estrutura, conexão e comandos SQL.	A ferramenta cria o banco, tabelas, conexões e comandos SQL automaticamente.
Documentação	Dispõe de livros, inúmeros sites, milhares de usuários e páginas oficiais.	Em sua maioria, é fornecida pelo fabricante.
Suporte	Apenas em livros, sites oficiais, sites da Web e programadores com	A fabricante dispõe de canais para auxiliar seus clientes.

	conhecimento.	
Mão de obra	Profissionais são ensinados em cursos, universidades, livros ou sites da Web.	Mão de obra escassa. Profissionais precisam ser treinados.
Portabilidade	Rodam em qualquer plataforma.	O GeneXus está disponível apenas para o sistema operacional Windows. O Scriptcase possui versões para todos os sistemas operacionais disponíveis.

Tabela 1 – Comparativo entre linguagens de programação e geradores de código.

6. Análise dos resultados

As linguagens de programação destacam-se como vantagens por sua portabilidade, dando ao programador a liberdade de trabalhar com qualquer plataforma do mercado. Destaca-se também que as linguagens não possuem custo e dispõe de vários profissionais e treinamentos disponíveis.

Todo o código escrito da aplicação é feita por membros da equipe, o que facilita futuras manutenções. Entretanto, como ponto negativo, os programadores precisam ter conhecimento da linguagem, buscar ajudas em meios alternativos e em grande maioria, desenvolver todas as funcionalidades do sistema.

Os geradores de código tem como desvantagens seu custo, a falta de mão de obra e dificuldade em manter o código gerado pela própria ferramenta. Porém, essas desvantagens são supridas pela facilidade em se aprender a utilizar a ferramenta.

Os geradores tendem a gerar todos os comandos de interação com o banco de dados e outros serviços, assim ganha-se tempo no desenvolvimento do sistema. Quando necessário, a equipe tem a sua disposição o suporte necessário para resolução de problemas ou dúvidas.

7. Conclusão

Cada empresa tem suas regras e padrões de trabalho, onde seus profissionais tendem a direcionar o desenvolvimento para tecnologias que tenham conhecimento e domínio. Uma mudança ou adoção de tecnologia diferente do habitual, em grande maioria dos casos, desagrada os profissionais. Porém, apontar qual é a melhor opção adotar não é uma tarefa simples, sendo ideal toda a equipe ser envolvida no processo de decisão.

Ao comparar geradores de código com linguagens de programação não é possível apontar de maneira superficial qual é a melhor opção a ser escolhida. Pode-se citar que um gerador tem um custo elevado, mas este é absorvido quando o ponto em questão é a oferta de suporte e treinamento, inclusive na adoção da ferramenta. O mesmo não ocorre com as linguagens, estas em sua maioria, sem custos de aquisição, mas que não dispõem de uma empresa de suporte e treinamento à disposição.

Possivelmente a forma mais correta de definir a melhor tecnologia, seria colocar a mesma equipe para desenvolver um software utilizando uma linguagem de programação e um gerador de código, de forma comparativa, utilizando a mesma linguagem de programação. Poderiam ser definidas métricas para mensurar tempo e qualidade, mas há outros aspectos que não podem ser avaliados, como máquinas, momento emocional da equipe, entre outros.

Conclui-se que cada equipe deve definir qual tecnologia adotar segundo suas observações e experiências. Entretanto, sempre que possível, é interessante que a equipe discuta o desempenho do projeto com a ferramenta utilizada, levantando aspectos positivos e negativos, buscando sempre melhorar o andamento dos processos, seja com a própria tecnologia ou na busca de uma nova opção.

Embora o presente estudo seja um comparativo entre linguagens de programação e ferramentas geradoras de código, como limitações da pesquisa, recomenda-se cautela na generalização dos resultados pois foram utilizadas apenas duas linguagens dentre milhares existentes e apenas duas ferramentas disponíveis no mercado, pois foram utilizados como base para o comparativo, o uso destas em empresas e universidades da região.

Como trabalhos futuros, recomenda-se o uso de métricas qualitativas e quantitativas para mensurar a criação de um artefato, tanto na linguagem quanto no gerador de código, utilizando os mesmos recursos computacionais e humanos para ambos de forma que se possa apontar qual a melhor alternativa considerando valores numéricos como resultante.

Referências

GENEXUS. Disponível em: <<http://www.genexus.com>>. Acesso em: 02 de Maio de 2016.

GERHARDT, F. **Integration of Programming Environments for Platform Migration**. 2003. 128 f. Dissertação (Mestrado) - Eberhard Karls Universität Tübingen, Tübingen, 2003.

HUNT, A.; THOMAS, D. **The pragmatic programmer**: from journeyman to master. Estados Unidos: Artmed, 1999.

Martins, J. C. C. **Técnicas para gerenciamento de projetos de software**. Rio de Janeiro: Brasport, 2007.

NAIM, R.; et.al. **Comparative Studies of 10 Programming Languages within 10 Diverse Criteria**. Artigo publicado no COMP, Montreal-QC, Canadá. Montreal: 2010. Disponível em: <https://arxiv.org/pdf/1008.3561.pdf>. Acesso em: 05 mai. 2016.

O'KEEFFE, J. **Análise de fatores de impacto no erro de estimativa de esforço e de duração em projetos de software**. 2012. 94 f. Dissertação (Mestrado) - Curso De Faculdade De Administração, Contabilidade e Economia Mestrado em Administração e Negócios, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2012.

PAQUET, J.; MOKHOV, S. A. **Comparative Studies of Programming Languages**. Course Lecture Notes, Montreal, v. 6, p. 7-61, 2010.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. 3. ed. Rio de Janeiro: Brasport, 2005.

SCRIPTCASE. Disponível em: <<http://www.scriptcase.com.br>>. Acesso em: 02 de Maio de 2016.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 9. ed. Tradução de Eduardo Kessler Piveta. Porto Alegre: Bookman, 2011.

TUCKER, A. B.; NOONAN, R. **Programming languages: principles and paradigms**. 2.ed. Estados Unidos: McGraw-Hill: 2007.

VAREJÃO, F. M.. **Linguagem de programação: conceitos e técnicas**. Rio de Janeiro: Elsevier: 2004.

VIRTUOSO, G.; COUTO MAIOR JUNIOR, M.; MARTINS, P. **Propriedades Desejáveis a uma Linguagem de Programação: Uma Análise Comparativa entre as Linguagens C, C++ e Java**. In: I Congresso Sul Catarinense de Computação, Criciúma-SC. Criciúma: 2005. Disponível em: <http://periodicos.unesc.net/sulcomp/article/viewFile/796/747>. Acesso em: 05 mai. 2016

DOCKER – APRESENTAÇÃO DA FERRAMENTA

Dan Lucio Prada¹, Natalia Kelim Thiel², Rodrigo Cachoeira³, Wesley dos Reis Bezerra⁴

¹ Graduando Bacharelado em Ciência da Computação – IFC – Rio do Sul

² Graduando Bacharelado em Ciência da Computação – IFC – Rio do Sul

³ Graduando Bacharelado em Ciência da Computação – IFC – Rio do Sul

⁴ Professor Bacharelado em Ciência da Computação – IFC – Rio do Sul

dlprada96@gmail.com, natalia.kthiel@gmail.com,

rodrigocachoeira11@gmail.com, wesley.bezerra@ifc.edu.br

Abstract. *In this article we have as our main objective the presentation of a tool called Docker. For this we use theoretical approaches presenting explanatory concepts about the tool, as well as practical examples in the everyday use of developers and system administrators. We present qualitative results referring the tool and its history compared to the market with the development of environment virtualization technology. Finally, we conclude its importance as a tool to help application managers around the world.*

Key-words: *Docker; Container; Virtualization; Docker Hub; Creating a Development environment.*

Resumo. *Neste artigo possuímos como principal objetivo a apresentação de uma ferramenta chamada Docker. Para isso utilizamos tanto abordagens teóricas apresentando conceitos explicativos sobre a ferramenta, como também exemplificações práticas no uso cotidiano de desenvolvedores e administradores de sistemas. Apresentamos resultados qualitativos referente a ferramenta e seu histórico em comparação ao mercado com o desenvolvimento da tecnologia de virtualização de ambientes. Por fim, concluímos a sua importância como ferramenta de auxílio aos gestores de aplicações em todo o mundo.*

Palavras-chave: *Docker; Container; Virtualização; Docker Hub; Criando um Ambiente de Desenvolvimento.*

1. Introdução

Com o avanço computacional e a criação de diferentes ferramentas focadas no desenvolvimento de software, torna-se visível a necessidade da utilização de metodologias que permitam a padronização atrelada a escalabilidade na criação de aplicativos.

Observando essa necessidade, a empresa Docker Inc, desenvolveu uma forma elegante de integrar ambientes de desenvolvimento distintos em um mesmo ecossistema, através da utilização de *containers*. Trazendo consigo metodologias que

auxiliam desenvolvedores e administradores de ambientes computacionais a gerenciarem com extrema facilidade seu projeto, fornecendo assim uma maior escalabilidade as aplicações e mantendo a capacidade de manutenção em um ambiente totalmente padronizado, além de fornecer uma forma simples de organizar o versionamento de diferentes projetos de software.

Docker é uma plataforma Open-Source escrita em Go³, multiplataforma e com extensa documentação de suas funcionalidades. Atualmente estima-se que o Docker está presente em mais de 70 países e com mais de 8 bilhões de *containers* baixados, sendo considerado por muitos o *software* mais completo do mercado de desenvolvimento, no que diz respeito ao encapsulamento de ambientes de produção de *software*.

2. Docker

Segundo Taurion (2009, p.16), *Cloud Computing*, é a entrega da computação como um serviço e não como um produto, onde recursos são compartilhados, *software* e informações são fornecidas, permitindo o acesso de qualquer computador, *tablet* ou celular que possua conexão com a internet.

O conceito de PaaS está vinculado ao uso de ferramentas e de desenvolvimento de software oferecidas por provedores de serviços, onde os desenvolvedores criam as aplicações e as desenvolvem utilizando a internet como meio de acesso. Neste caso o provedor oferta a plataforma de desenvolvimento. PaaS está fortemente atrelado a utilizar produtos de terceiros, a responsabilidade de armazenagem de informações e execução de aplicativos fica atribuída como função da plataforma. (VERAS, 2012, p. 169)

Inicialmente o Docker foi inventado para que a Dor Cloud pudesse suportar de forma mais simples a gerência de seu PaaS, onde desenvolvedores poderiam fazer *deploy*⁴ de suas aplicações de uma maneira similar ao Heroku⁵, mas, em vez de máquinas virtuais debaixo dos panos, tudo rodaria em *containers* Linux. (SILVA, 2016, p. 23)

Por ser um projeto *open source*, qualquer pessoa pode visualizar o código e contribuir com melhorias para o Docker. Isso traz maior transparência e faz com que correções de *bugs* e melhorias aconteçam bem mais rápido se comparado com um *software* proprietário. Além dos benefícios citados anteriormente, o Docker possibilita ao desenvolvedor testar suas aplicações em inúmeras configurações de ambientes, além de permitir que sua interação com o computador seja feita por um S.O (Sistema Operacional). e seu ambiente de desenvolvimento esteja centralizado em outro Sistema Operacional fazendo com que tenha inúmeras possibilidades.

Quando o Docker 1.0 foi lançado e anunciado que estava pronto para produção, empresas como Spotify⁶ já utilizavam em grande escala; logo AWS e Google começaram a oferecer suporte a

3 Linguagem de programação de alto desempenho desenvolvida dentro do Google, que facilita a criação e administração de ambientes isolados.

4 Colocar o projeto em produção, ou seja, permitir que o cliente final utilize-o como produto validado.

5 É uma plataforma de serviço em nuvem suportando várias linguagens de programação.

6 Serviço de música comercial em *streaming*.

Docker em suas nuvens. Outra gigante que aderiu esta tecnologia foi a Red Hat⁷, que se tornou uma das principais parceiras do Docker, inclusive incorporando-o ao OpenShift⁸.

2.1. Containers

Vitalino e Castro (2016, p. 3) afirmam que *container* é, em português, o agrupamento de uma aplicação junto com suas dependências, que compartilham o *kernel* do sistema operacional do *host*, ou seja, da máquina (seja virtual ou física) onde está rodando. *Containers* são bem similares às máquinas virtuais, porém mais leves e mais integrados ao sistema operacional da máquina *host*, uma vez que compartilha o seu *kernel*, o que proporciona melhor desempenho por conta do gerenciamento único dos recursos.

Na maioria dos casos, a imagem de um *container* é bastante compactada, havendo somente o necessário para o funcionamento da aplicação, que, quando em execução, possui um pequeno *overhead* se comparada à mesma aplicação rodando nativamente no sistema operacional, grande parte disso por conta do compartilhamento dos recursos com a máquina *host*.

Quando se está utilizando máquina virtual, ocorre uma emulação de um novo sistema operacional e todo o seu hardware utiliza recursos da máquina *host*, o que não ocorre quando utiliza-se *containers*, pois os recursos são compartilhados. O maior benefício do que foi apresentado é a capacidade de rodar mais *containers* em um único *host*, se comparado com a quantidade que se conseguiria com máquinas virtuais.

É possível realizar o empacotamento de uma aplicação ou ambiente inteiro dentro de um *container*, e a partir desse momento o ambiente torna-se portátil para qualquer outro *host* que contenha o Docker instalado.

Aplicando os processos listados, o tempo de *deploy* de uma aplicação é drasticamente reduzido pois não há necessidade de ajustes ao ambiente para o correto funcionamento do serviço e podendo replicá-lo quantas vezes forem necessárias.

2.2. Ecossistema Docker

Sabe-se que o desenvolvimento de um software apresenta inúmeros obstáculos durante seu ciclo de construção e, eventualmente, para a produção. Além do trabalho inegável de desenvolvimento do software para permitir seu pleno funcionamento em diferentes ambientes, também ocorrem problemas com rastreamentos de dependências e escalabilidade, referente a própria aplicação e atualização de componentes individuais, sem afetar a aplicação como um todo.

Segundo Ellingwood (2015) a containerização e isolamento surgem visando melhorar o desenvolvimento de uma aplicação, apresentando conceitos que não são novos no mundo da computação. Alguns sistemas operacionais *Unix-like* alavancaram tecnologias maduras de

⁷ Um das distribuições de linux mais famosas no Estados Unidos.

⁸ É uma plataforma de aplicativos em *containers* que oferece as tecnologias Docker.

containerização por mais de uma década. Em 2008, a metodologia deu seus primeiros passos no Linux LXC, o Sistema Operacional combinou o uso de *cgroups* do *kernel*, permitindo o isolamento na utilização dos recursos e separando-os em grupos de forma que eles não consigam interagir entre si.

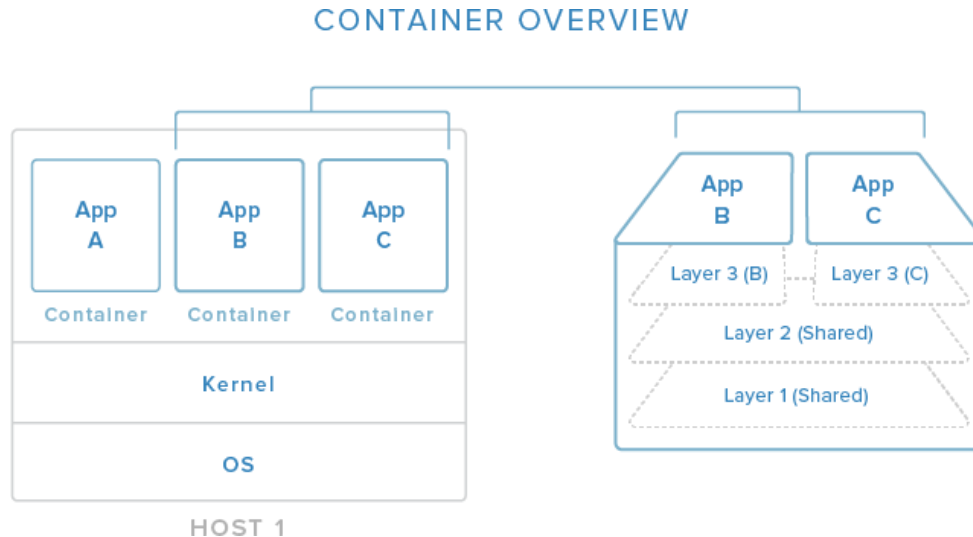


Figura 1 – Containers

Na Figura 1, é possível visualizar de forma simplificada como os *containers* se relacionam com o sistema *host*. Os *containers* isolam aplicações individuais e utilizam recursos do sistema operacional que foram abstraídos pelo Docker. Na direita, é possível ver que os *containers* podem ser construídos por “camadas”, com vários *containers* compartilhando camadas subjacentes, diminuindo o uso de recursos.

2.2.1. Padronização de Ambientes

Como mencionado anteriormente, a metodologia de *containers* facilita o desenvolvimento de aplicações utilizando formas de padronização de ambientes, dando tranquilidade a aplicação, pois esta não deve dar relevância, na maioria dos casos, aos recursos arquiteturais de seu *host*. Simplificando premissas de desenvolvimento sobre o ambiente operacional. Da mesma forma, para o *host*, todo *container* é uma caixa-preta que não deve atribuir importância para o que há dentro.

2.2.2. Escalabilidade

Além da padronização também é possível observar o aumento da escalabilidade de um projeto. Um desenvolvedor pode executar alguns *containers* em sua estação de trabalho, enquanto

este sistema pode ser escalado horizontalmente em uma área de preparação ou teste. Quando os *containers* entram em produção, eles podem escalar novamente.

2.2.3. Versionamento

Os *containers* permitem que um desenvolvedor empacote uma aplicação ou um componente de aplicação juntamente com todas as suas dependências como uma unidade. Isso torna viável o gerenciamento de dependências e também simplifica o gerenciamento de versões da aplicação, portanto é possível ter versões de *containers* que utilizem determinadas aplicações que não são utilizadas por outras versões do *container*.

2.2.4. Camadas Compartilhadas

Os *containers* são leves no sentido de que eles estão alocados em camadas. Se múltiplos *containers* estão baseados na mesma camada, eles podem compartilhar a camada subjacente sem duplicação, levando a um uso de espaço em disco mínimo para as imagens posteriores.

3. Apresentação das Ferramentas

Para apresentação da ferramenta Docker utilizou-se um ambiente Linux, utilizando a distro Ubuntu em sua versão 16.10, 64bits.

3.1. Imagens e *containers*

Antes de prosseguir com os demais comandos é necessário apresentar os conceitos de imagem e *container* de forma prática no Docker.

Para criar uma virtualização de qualquer aplicação é necessário uma imagem, que consiste em representar as configurações do ambiente a ser virtualizado, comparado com uma máquina virtual, a imagem seria o arquivo ISO e o *container* seria o ambiente virtualizado a partir da imagem.

O Docker possui um repositório de imagens chamado Docker Hub, acessando <https://hub.docker.com/explore> é possível visualizar todas as imagens públicas para *download* gratuitamente.

Através destas imagens é possível criar um *container* que executará a aplicação. Uma imagem pode ser derivada em inúmeros *containers*, cada qual com seu próprio ciclo, será nos *containers* que o seu ambiente será executado. Por exemplo, é possível ter uma imagem do sistema operacional CentOS⁹ e a partir dela criar várias virtualizações (*containers*) que irão rodar o sistema operacional. O Docker fica responsável por armazenar os *containers*, estes que podem ser acessados de qualquer lugar do ambiente.

9

Abreviação de *Community ENTERprise Operating System*, é uma distribuição linux de classe corporativa.

3.2. Execução de um *container*

Para realizar a execução de um *container* deve-se primeiro baixar a imagem, para esse exemplo será usado a imagem do sistema operacional Ubuntu.

1	<code>docker pull ubuntu</code>
2	<code>docker images</code>
3	<code>docker run -it ubuntu</code>

Tabela 1 – Comandos de criação de um *container*

Executando no terminal o comando especificado na linha 1, a imagem do S.O. será baixada para o repositório local de imagens do Docker. Após isso para visualizar as imagens disponíveis no repositório local do Docker basta executar o comando na linha 2.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
222	latest	59014023f0ab	43 hours ago	1.01GB
ubuntu	dev	222242da7631	2 days ago	749MB
ubuntu	latest	b4b32724200e	2 days ago	188MB
mysql	latest	e799c7f9ae9c	8 days ago	407MB
ubuntu	<none>	f7b3f317ec73	3 weeks ago	117MB

Figura 2 –

Listar as Imagens

A Figura 2 representa a saída no terminal após executar os comandos listados. Na primeira coluna é apresentado o nome da imagem, na segunda coluna uma *tag*¹⁰, por exemplo, a imagem do Ubuntu pode possuir tags indicando suas versões, Ubuntu Tag: 16.04, Ubuntu Tag: 17.04. Na terceira coluna é apresentado o identificador da imagem, no Docker é possível trabalhar tanto com o nome da imagem como também seu identificador. E por último a data de criação e o tamanho da imagem.

Para criar um *container* da imagem, ou seja, visualizá-la deve-se utilizar o comando especificado na 3 linha da Tabela 1.

Por meio deste comando um *container* da imagem será executada no terminal. Analisando o comando executado tem-se que o termo *run* dispara a intenção de se criar um *container* de uma imagem, logo a seguir, os parâmetros *-it* indicam que será executado de forma interativa, ou seja, o terminal da imagem, caso seja um sistema operacional será aberto no console atual e por fim o nome da imagem que será executada. Após o comando, deverá ser aberto o *console* com uma instalação padrão do Ubuntu. Seguindo esses passos o sistema operacional já está virtualizado em um Docker *Container*.

3.3. Comandos Úteis

Para gerenciar os *containers* e imagens presentes no repositório local do Docker alguns comandos são importantes, como por exemplo visualizar quais *containers* você possui instanciados,

¹⁰ São etiquetas que ajudam a organizar informações.

lembrando que um *container* pode ter dois *status*, ou ele está em execução ou parado. Segue uma tabela explicativa com alguns comandos úteis.

Comando	Explicação
<code>docker ps</code>	Lista os Containers em execução.
<code>docker ps-a</code>	Lista os Containers em execução e parados.
<code>docker start {ID Name}</code>	Inicializa um container pelo seu id ou nome.
<code>docker stop {ID Name}</code>	Para um container pelo seu id ou nome.
<code>docker exec-it {ID Name}</code>	Entra em modo iterativo em um container já inicializado.

Tabela 2 – Comandos Básicos do Docker

3.4. Persistência do *container*

É importante ressaltar um dos detalhes do Docker: ao instanciar um *container*, este pegará todos os dados salvos na imagem de sua origem, programas instalados, dados salvos, porém após utilizar um *container* e pará-lo ou até mesmo excluí-lo, todas as informações contidas nele serão removidas, não é realizada persistência durante o tempo de execução de um *container*. Uma forma simples de resolver este impasse é, ao iniciar e execução de um *container* deve-se realizar um vínculo, uma sincronização de pastas entre o HD local da máquina, na qual o docker está sendo executado e a virtualização do ambiente que será instanciada, logo qualquer alteração em um caminho específico dentro de ambiente de virtualização será replicado para o HD local, solucionando o problema de perda de dados caso o ambiente seja removido.

1	<code>docker run-it-v /nome/pasta/local/sync:/nome/pasta/virtualizada ubuntu</code>
2	<code>docker commit {ID Name} {ID Name : version}</code>

Tabela 3 – Sincronização do *container* ubuntu e outros

Para isso sincronizou-se as pastas utilizando a imagem do Ubuntu como modelo, através do comando da 1 linha apresentando na tabela 3.

A primeira parte do comando já é conhecida, o diferencial é o parâmetro-v adicionado, este representa o vínculo entre pastas, os valores passados após ele são divididos em duas partes, a primeira indica o local na máquina física que os dados provenientes da virtualização serão salvos, a segunda parte, separada por ':', define qual pasta da máquina virtualizada será enviada para máquina física.

Anteriormente foi mencionado sobre o versionamento de imagens, em que uma imagem pode possuir inúmeras versões, cada qual com suas peculiaridades. Para realizar uma atualização de um *container*, é possível atualizar a imagem original, criar versões dela ou até mesmo criar outra

versão, funcionalidades que podem ser feitas utilizando o método *commit* do docker, o qual persistirá toda e qualquer modificação de um *container* em uma imagem.

Analisando a linha 2 da tabela 3, o primeiro parâmetro contém o identificador do *container* a ser salvo, como segundo parâmetro o nome da imagem seguido ou não pela sua versão, caso não seja informado o nome de uma imagem pré-existente uma nova imagem será criada com o *container* informado.

3.5. Docker Hub

O Docker possui as imagens disponíveis em seu repositório local na máquina que o está executando, porém também possui um repositório remoto, o qual já foi utilizado anteriormente, para baixar o sistema operacional Ubuntu. Sua principal função é armazenar imagens para que qualquer um, caso possa realizar o download, caso essas imagens sejam salvas como públicas. Para buscar imagens no repositório remoto é possível utilizar o terminal por meio do comando na linha 1 da tabela 4.

1	docker search {Name}
---	----------------------

Tabela 4 – Busca por imagens

Como parâmetro deve-se utilizar o nome do repositório que está sendo procurado. Para realizar o *download* do repositório deve-se trocar a palavra *search* para *pull* e caso seja um *upload* para o repositório remoto utiliza-se a palavra *commit*.

3.6. Criando um Ambiente de Desenvolvimento

Para melhor entendimento do ambiente Docker, será realiza a configuração de um ambiente de desenvolvimento *web* utilizando *softwares* como *apache*, *php* e *mysql*. Para isso será criado dois *containers*, um que possuirá a linguagem de programação e seu interpretador e o outro container será responsável apenas por manter o banco de dados.

Inicialmente deve-se baixar as duas imagens para o seu repositório local como especificado nas linhas 1 e 2 da tabela 5.

1	docker pull ubuntu
2	docker pull mysql
3	docker run -it ubuntu
4	apt-get update
5	apt-get install apache2
6	apt-get install php

7	exit
8	docker ps -a
9	docker commit {ID Name} ubuntu:dev-version
10	docker run -it -v /minha/pasta/db:/var/lib/mysql --name mysql-server -p 3308:3306 -e MYSQL_ROOT_PASSWORD=admin mysql
11	docker run -it -p 8081:80 -v /minha/pasta/dados:/var/www/html/site ubuntu:dev-version

Tabela 5 – Comandos de Instalação do ambiente

Após o *download* das imagens será necessário configurar o ambiente de desenvolvimento que será criado a partir da imagem Ubuntu. Para isso basta executar as linhas 3, 4, 5 e 6 na tabela 5.

Após as instalações o *container* que possui o interpretador e a linguagem que será utilizada estará pronta, porém deverá ser efetuado uma última etapa, realizar a persistência da virtualização em uma imagem para utilizá-la novamente, como comentado, um container virtualizado não realiza persistências automaticamente. Para persistir as novas atualizações deve-se sair do terminal e atualizar a versão da imagem como mostrado nas linhas 7, 8 e 9 da tabela 5.

Foi realizado o comando para visualizar os *containers* disponíveis, pois para realizar um *commit*, é necessário saber qual o identificador do *container*, para isso a listagem dos *containers*. O servidor de desenvolvimento foi salvo na imagem Ubuntu em sua versão *dev-version*.

Agora os ambientes estão configurados e prontos para uso, para apresentá-los deve-se realizar uma vinculação dos arquivos virtualizados com uma pasta física para não perdê-los, os comandos listados na tabela 5, nas linhas 10 e 11 mostram a virtualização das duas imagens.

Os dois *containers* estão sendo executados, foi sincronizado os bancos de dados mysql no HD físico como pode-se visualizar no comando, e a mesma coisa foi feita com o *site* dentro do servidor de desenvolvimento. Para visualizá-los funcionando, deve-se acessar o seguinte IP no *browser* da máquina local 172.17.0.1, deverá ser listado as configurações do PHP. Para listar o banco de dados mysql, em qualquer aplicativo de conexão de banco de dados realize a conexão utilizando o ip 172.17.0.2 com a porta 3308.

4. Resultados e Discussões

É visível que o Docker vem ao mercado com uma proposta muito benéfica a linha de produção de software, tendo um salto grande em popularidade por conta das funcionalidades disponíveis. Porém a sua adoção em ambientes reais ainda caminha em passos pequenos, isso se dá principalmente por conta da necessidade de treinamento e conhecimento sobre a ferramenta, além de levantar alguns questionamentos sobre o custo e benefícios que a ferramenta pode trazer para o desenvolvimento.

A fim de debater estes questionamentos, foram realizados alguns testes em máquinas sobre diferentes circunstâncias e foram desenvolvidos projetos simples com o objetivo de analisar suas

funcionalidades, notou-se algumas características que demonstram o quão benéfica pode se tornar sua adoção, por exemplo, sua flexibilidade, escalabilidade, portabilidade, praticidade, dentre outros.

Analisando os resultados obtidos a partir do encapsulamento das aplicações, notou-se a simplicidade em executar um mesmo projeto de desenvolvimento em ambientes distintos, porém com os mesmos resultados. Para isso, executou-se a aplicação em um ambiente local de desenvolvimento e em um servidor mais robusto, gerando os mesmos resultados, isso auxilia muito no ciclo de desenvolvimento, pois os *containers* onde serão desenvolvidos os projetos serão os mesmos no servidor, portanto reduz-se problemas repentinos em tempo de execução. Além disso, caso a aplicação necessite de recursos externos para seu funcionamento, basta empacotá-las dentro da imagem e disponibilizá-la com a aplicação, excluindo assim a necessidade de gerar uma extensa documentação sobre a configuração de ambientes de desenvolvimento compatíveis com o projeto, vale ressaltar que as imagens e *containers* ainda possibilitam maior agilidade maior no desenvolvimento contínuo de softwares, simplificando consideravelmente atualizações, pois o Docker se responsabiliza por alterar na imagem apenas as modificações realizadas.

Sua adoção vem aos poucos no mercado, e mesmo com sua implementação algumas empresas tendem a realizar tal integração com certa cautela, mesmo grandes corporações tais como Google, Spotify e Facebook. Ainda realiza-se a sua instalação dentro de máquinas virtuais, entretanto sabe-se que a adoção do Docker exclui a necessidade de tal estratégia. Todavia há uma grande preocupação em questões de disponibilidade e segurança sobre os projetos, apenas a utilização dos containers é insatisfatória, pois algum container pode simplesmente falhar e limitar o acesso aos demais ao kernel.

Em todos os testes realizados foram obtidos resultados satisfatórios, auxiliando e facilitando o desenvolvimento de projetos em equipe, excluindo esforços desnecessários, pois o Docker disponibiliza um local para publicação de imagens de serviços, assim como o Github existe toda uma comunidade que inclui até mesmo desenvolvedoras de serviços como Oracle, que disponibilizam imagens de serviços prontos e já integrados utilizando as melhores práticas e configurações, mas isso não significa que o desenvolver ficará engessado a essas configurações pré-definidas, uma vez que é possível realizar qualquer customização por parte do utilizador, sendo assim resultando em apenas benefícios e redução de trabalho desnecessário.

5. Conclusão

O Docker fornece os blocos construtivos fundamentais necessários às implantações de *containers* distribuídos. Através do empacotamento dos componentes, das aplicações em seus próprios *containers* e da sua escalabilidade horizontal torna-se um simples processo de lançar ou desligar múltiplas instâncias de cada componente.

Sua adoção agiliza e facilita no desenvolvimento de projetos em equipe, além de auxiliar o desenvolvimento de softwares continuamente, uma vez que permite o envio da imagem alterada ao servidor e aos clientes. Existem alguns fatores que ainda influenciam na sua adoção em massa, porém sendo uma tecnologia recente ela vem demonstrando um grande amadurecimento ao longo do tempo. Vale ressaltar que o Docker não vem com o objetivo de solucionar todos os problemas existentes e sim como mais uma ferramenta que auxilia desenvolvedores em alguns dos seus

problemas corriqueiros no seu cotidiano de produção, tentando ao máximo trazer benefícios para as empresas que adotarem, além de facilitar o desenvolvimento distribuído de softwares.

Concluindo, o Docker é uma ferramenta leve e *developer-centric* que permite executar “máquinas virtuais” instantaneamente em comandos de linha, compartilhando recursos do sistema *guest*. Dentro de seu universo de ação (focado na aplicação), ele se apresenta como uma ótima ferramenta. Todavia, fora desse universo ainda tem poucas aplicações, por essa razão é possível dizer que ele não é a evolução da virtualização, apenas uma alternativa que ainda deve amadurecer ao decorrer do tempo.

Referências

- ELLINGWOOD, Justin. **O ecossistema do Docker**. 2016. Disponível em: <<https://www.digitalocean.com/community/tutorials/o-ecossistema-do-docker-uma-introducao-aos-componentes-comuns-pt>>, Acesso em: 2017 de mai, 2017.
- SILVA, W. F. **Aprendendo Docker**. 1. ed. São Paulo, SP, 2016.
- TAURION, C. **Clouding Computing, Computação em Nuvem**. 1. ed. Rio de Janeiro, RJ. 2009.
- VITALINO, J. F. N; CASTRO M. A. N. **Descomplicando o Docker**. 1. ed. Rio de Janeiro, RJ, 2016.
- VERAS, F. **Clouding Computing: Nova Arquitetura da TI**. 1. ed. Rio de Janeiro, RJ, 2013.

PLAYER MR: REPRODUTOR DE ÁUDIO COM LEGENDAS

Marcos Momm¹, Rodrigo de Moraes¹, Rodrigo Curvêllo²

¹Estudante de Bacharel em Ciência da Computação

²Professor e Mestre em Engenharia Elétrica

marcos.momm@hotmail.com.br, moraesrodrigo6@gmail.com.br,

rodrigo.curvello@ifc-riodosul.edu.br

Abstract. *The development of a music player demonstrates the need to use concurrent programming, which Java language supports development. With the use of JavaFX for the interface and the use of Java threads to control the graphical part, the execution of the song and manipulation of the legend for the user, it was possible to create the MR music player.*

Key-words: *Player; Java; Thread.*

Resumo. *O desenvolvimento de um player de música demonstra a necessidade da utilização da programação concorrente, qual a linguagem Java dá suporte ao desenvolvimento. Com o uso do JavaFX para a interface e o uso de threads do Java para controlar a parte gráfica, a execução da música e manipulação da legenda para o usuário, foi possível a criação do player de música MR.*

Palavras-chave: *Player; Java; Thread.*

1. Introdução

Nos sistemas operacionais tradicionais, cada processo possui um espaço de endereçamento e um único fluxo de controle. Entretanto, existem situações em que é preferível, ou mesmo necessária, a execução praticamente paralela de vários fluxos de controle no mesmo espaço de endereçamento, (TANENBAUM; WOODHULL, 2008). Esses fluxos de controle são conhecidos como *threads*. Em um programa *multithread* há várias *threads* rodando concorrentemente com um único espaço de memória e cada uma com sua própria sequência de instruções

O desenvolvimento do *player* de música MR, onde, MR são as letras iniciais de Marcos e Rodrigo, os autores desse trabalho, foi utilizado a linguagem de programação Java com o seu suporte para programação concorrente e de toda a lógica do player de música, JavaFX responsável por toda a parte gráfica e layout e disposição dos botão na interface do *player* de música e foi utilizado o formato da estrutura do Json, usado nos arquivos de legenda.

2. Fundamentação Teórica

Para a criação do *player* de música MR, foi utilizado a linguagem de programação Java, que tem a possibilidade da programação concorrente com o uso das APIs e que é responsável por todo o funcionamento do *player* de música, na qual, cada botão que o usuário clica é a linguagem Java rodando por trás do *player* para a realização do evento do botão em que o usuário clicou.

Na interface gráfica do *player* de música MR foi utilizado o JavaFX, segundo Clarke et. al. (2010), é uma família de produtos desenvolvidos na Sun Microsystems. JavaFX é uma plataforma que inclui uma linguagem de script declarativa e de alto desempenho para oferecer e criar interfaces gráficas.

O foco primário do JavaFX é tornar o desenvolvimento de interface gráfica do usuário fácil e adotar características mais atraentes como efeitos visuais, som e animação. O JavaFX inclui uma *framework* pronto para suportar componentes gráficos e facilmente incluir características de multimídia. Usando a plataforma Java como seu núcleo, o JavaFX funciona continuamente como a plataforma Java e pode facilmente alavancar seu código Java existente. Isso também permite que o JavaFX implemente a capacidade “escreva uma vez, execute em qualquer lugar” oferecido pela plataforma Java, (CLARKE et. al., 2010).

Para a criação das legendas foi utilizada o padrão do Json (*JavaScript Object Notation*). Json é um modelo para armazenamento e transmissão de informações no formato texto. Apesar de ser simples, tem sido bastante utilizado por aplicações *Web* devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o *parsing* dessas informações. Isto explica o fato de o Json ter sido adotado por empresas como *Google* e *Yahoo*, cujas aplicações precisam transmitir grandes volumes de dados, (CÔRREA, 2017).

A ideia utilizada no Json para representa informações é simples, para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado. Seguindo esse padrão é que foram montados os arquivos de legenda, em que os rótulos utilizados nos arquivos de legenda são: sequência, tempo_inicial, tempo_final e o texto, que é a parte da letra da música cantada entre o tempo inicial e o tempo final.

A linguagem de programação Java disponibiliza a concorrência através da linguagem e das APIs (*Application Programming Interface* - Interface de Programação de Aplicações). Cada *thread* tem sua própria pilha de chamadas de método e seu próprio contador de programa, o que possibilita a execução simultânea com outras *threads* e o compartilhamento de recursos no nível do aplicativo. Diferentemente de linguagens que não possuem capacidades de *multithreading* integradas, as quais devem realizar chamadas não portáveis para primitivos de *multithreading* do sistema operacional, no Java os primitivos de *multithreading* estão incluídos como parte da própria linguagem e de suas bibliotecas. Esta distinção facilita a portabilidade entre plataformas com a manipulação de *threads*, em relação a outras linguagens, (DEITEL; DEITEL, 2010).

A classe *Thread* no Java, já é algo nativo na própria linguagem, podendo ser estendida ou implementada através da classe *Runnable* em qualquer classe criada. E através do método *Run* é onde o programador coloca as linhas de código ou chamada de métodos para que estes executem concorrentemente, conforme figura 1.

```

12  public class MinhaThread implements Runnable{
13
14      @Override
15      public void run() {
16
17      }
18  }

```

Figura 1 – Classe que implementa a interface *Runnable*. (Acervo do autor)

3. Desenvolvimento do player de música MR

O desenvolvimento do *player* se deu com intuito de proporcionar ao usuário, além da disponibilidade de reproduzir arquivos em formato mp3, as opções de cadastrar e salvar *playlists* de músicas e visualizar legendas de suas músicas.

Para o cadastro das *playlists*, basta utilizar o canto esquerdo da interface, onde as músicas podem ser selecionadas, salvas e editadas. Para salvar ou abrir uma *playlist*, pode ser utilizado o menu “Arquivo” da interface. Essas opções podem ser vistas na imagem da interface, apresentada na figura 2.

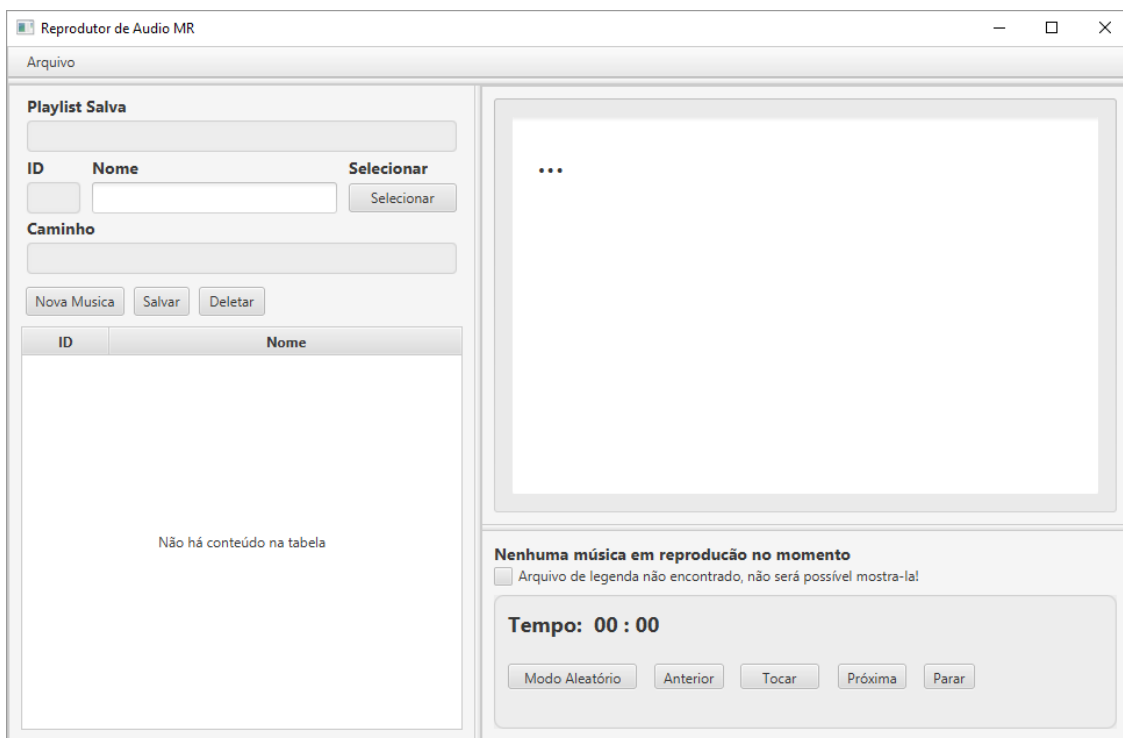


Figura 2 – Interface do Reprodutor de Áudio MR. (Acervo do autor)

No canto inferior direito, estão as informações da música que está sendo tocada atualmente, junto com os botões de opções (Modo Aleatório, Anterior, Tocar/Pausar, Próxima, Parar). Na parte superior direita, é mostrado uma área de texto que mostrará a legenda ao usuário.

Ao salvar uma *playlist*, o programa salva um arquivo com extensão “*plt*”, que também, é a única extensão de arquivo reconhecida na hora de se abrir uma *playlist*. Dentro deste arquivo, as músicas são salvas na estrutura de Json, seguindo o mesmo modelo da classe “Musica”, que pode ser observada no diagrama de classe junto ao Apêndice 1. Basicamente, em tempo de execução, uma *playlist* é manipulada através de uma lista de objetos do tipo “Musica” e a partir do momento do salvamento, esta lista de objetos, que é do tipo “*ObservableList*”, bastante utilizado junto ao JavaFX, é convertida em um Json. O processo inverso ocorre no momento que uma *playlist* é aberta, ou seja, ao abrir uma *playlist*, o programa lê o arquivo selecionado em busca de uma Json no mesmo formato de uma lista de objetos do tipo “Musica”, e assim realiza a conversão do Json para uma “*ObservableList*”.

As legendas das músicas, funcionam de forma semelhante a arquivos de legendas de vídeos, porém neste caso, no padrão de Json. O padrão de legenda utilizado pelo programa pode ser visto na figura 3.

```

1  {
2      "Arquivo": "PorcaVeia-LuzDoMeuRancho",
3      "letra": [
4          {
5              "sequencia": 1,
6              "tempo_inicial": "00:18",
7              "tempo_final": "00:21",
8              "texto": "Se às vezes me esqueço em função da vida,"
9          },
10         {
11             "sequencia": 2,
12             "tempo_inicial": "00:22",
13             "tempo_final": "00:26",
14             "texto": "de dizer que a vida pra mim é você,"
15         },
16         {
17             "sequencia": 3,
18             "tempo_inicial": "00:26",
19             "tempo_final": "00:30",
20             "texto": "te abanca comigo me faz um afago,"
21         }
22     ]
23 }

```

Figura 3 – Estrutura Json do arquivo de legendas. (Acervo do autor)

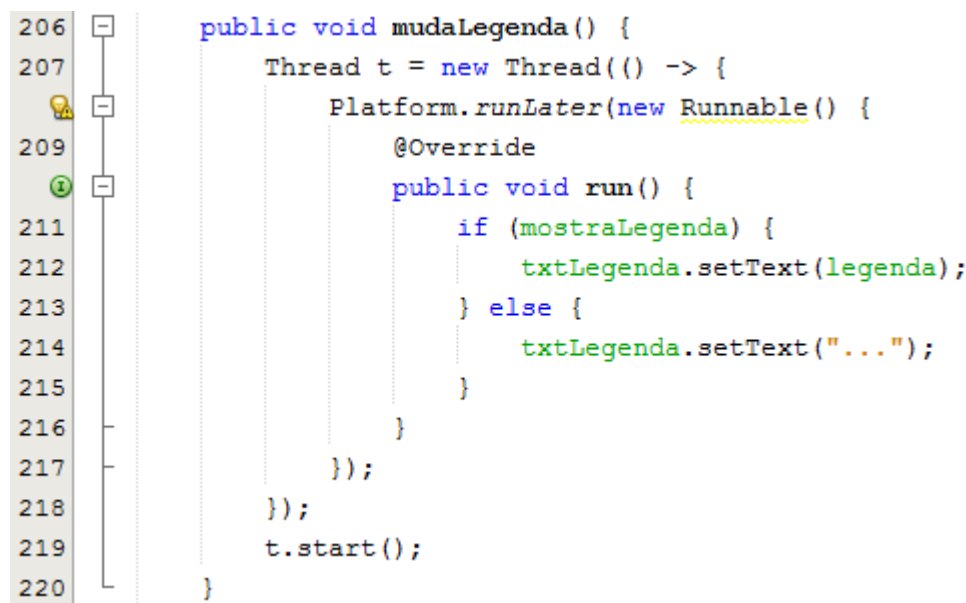
Como pôde ser acompanhado, o arquivo de legenda, armazena dois atributos, um deles contendo o nome do arquivo, e o outro atributo sendo uma lista de frases, que compõem a letra da música. Nesta lista de frases, tem-se um vetor de objetos, onde cada um destes objetos, representam um trecho da música. Neste objeto, além de termos o atributo “texto” temos também, outros 3 atributos. O “tempo_inicial” é referente ao tempo de início que a legenda aparecerá na tela, e o “tempo_final”, será o momento em que a frase desaparecerá da tela. O atributo “sequencia” serve apenas como controle na sequência das frases.

As legendas são salvas em arquivos com extensão “*lgd*” e deverão ser criadas manualmente pelo usuário que desejar que sua música seja legendada. O nome do arquivo, deverá ter o mesmo

nome do arquivo mp3, porém com a diferença que a extensão do arquivo será “*lgd*”. Com isso o programa entenderá automaticamente que aquela música possui legenda e a apresentará na tela.

Referente à programação concorrente, foram utilizadas duas *threads* principais, uma *thread* para a reprodução da música no java, utilizando uma biblioteca de terceiros atendendo pelo nome “*Player*”, e a outra *thread* controlando os componentes da tela, ou seja, a mudança do contador do tempo e das frases de legendas. Ao iniciar ou pausar uma música, as duas *threads* funcionam em conjunto para que a música e a legenda não saiam de sincronia.

Um problema encontrado ao utilizar as *threads* do java junto ao JavaFX foi que as *threads* do java não possuem acesso aos componentes gráficos do JavaFX. Com esse problema a *thread* que manipula os componentes da tela não poderia atualizar a interface para o usuário. Como solução, encontrou-se necessário a utilização de *threads* secundárias para a alteração de elementos gráficos na tela. Ou seja, quando se está executando uma *thread* java, para que se possa editar algum componente na tela, é necessário disparar uma *thread* do tipo “*runLater*” do JavaFX, para que não haja conflito e que o componente seja editado sem problemas na tela. Segue na figura 4, o código referente a função que modifica a legenda da música. Neste método, está sendo criada uma *thread* (linha 207) com método “*Platform.runLater*”. E logo após a declaração da *thread*, onde já está sendo chamada a execução dela (linha 219).



```

206 public void mudaLegenda() {
207     Thread t = new Thread(() -> {
208         Platform.runLater(new Runnable() {
209             @Override
210             public void run() {
211                 if (mostraLegenda) {
212                     txtLegenda.setText(legenda);
213                 } else {
214                     txtLegenda.setText("...");
215                 }
216             }
217         });
218     });
219     t.start();
220 }

```

Figura 4 – Função para alteração de legenda na tela. (Acervo do autor)

4. Conclusão

Este trabalho apresentou o desenvolvimento do *player* de música MR em que o usuário além de ouvir música tem a possibilidade de ter a legenda da música, que é mostrada ao mesmo tempo em que a música vai sendo executada.

Para que todos esses eventos ocorram, execução da música, legendas, se mostrou necessário uma linguagem de programação com suporte à programação concorrente, onde, a linguagem de programação Java tem o suporte para a programação concorrente. Na parte da interface foi utilizado o JavaFX, na criação do layout e disposição dos botões do *player* de música.

Na parte das legendas do *player* de música, que seria o diferencial, foi utilizado a estrutura Json, armazenando o tempo inicial e o tempo final e o texto que é a parte da letra da música cantado nesse intervalo de tempo.

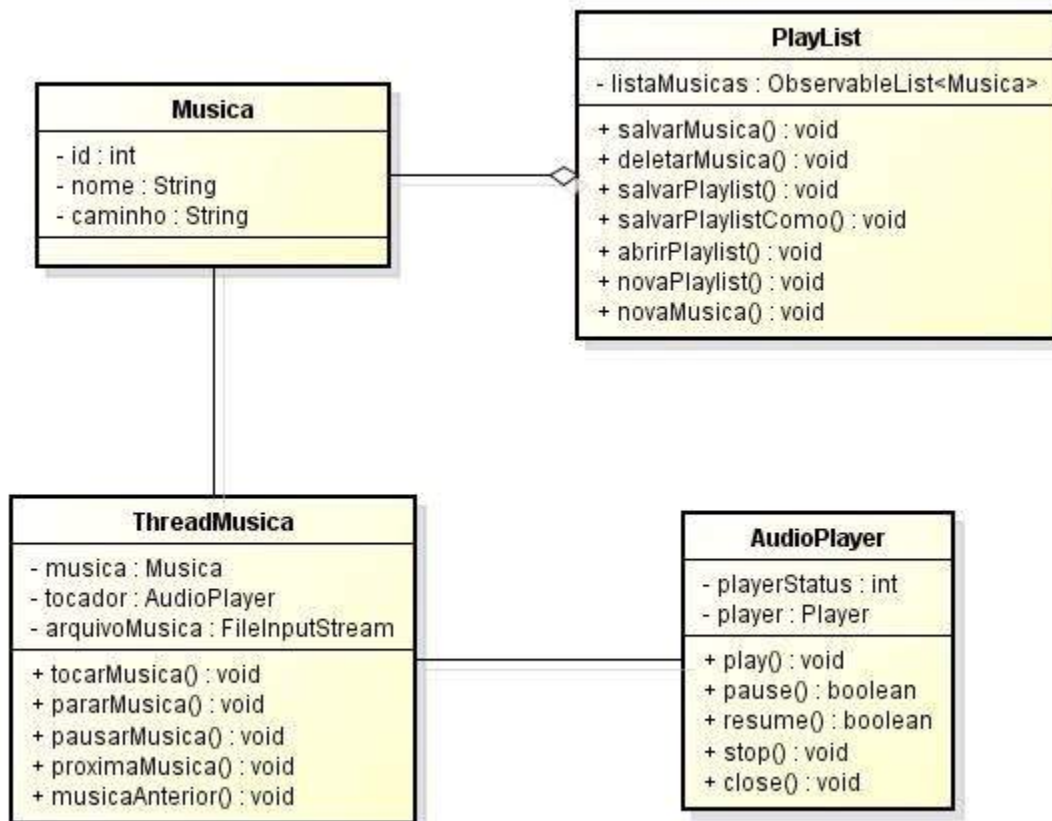
Referências

A. S. TANENBAUM AND A. S. WOODHULL, **Sistemas Operacionais, projeto e implementação**, 3 ed. Porto Alegre: Bookmann, 2008.

CÔRREA, Eduardo. **Introdução ao formato JSON**. Disponível em: <
<http://www.devmedia.com.br/introducao-ao-formato-json/25275> >. Acessado em: 19 mar. 2017 às 18h15min.

CLARKE, Jim; CONNORS, Jim; BRUNO, Eric. **Java FX: desenvolvimento de aplicações de internet ricas**. Rio de Janeiro: Alta Books, 2010. 325 p.

H. M. DEITEL AND P. J. DEITEL, **Java Como Programar**, 8 ed. São Paulo: Pearson Prentice Hall, 2010.

APÊNDICE I – DIAGRAMA DE CLASSE DO PLAYER MR (Acervo do autor)

JOGO DAS DERIVADAS

William Wessner¹, Rodrigo Curvêllo²

¹ Acadêmico do curso de Ciência da Computação do Instituto Federal Catarinense –
Campus Rio do Sul

² Professor do Instituto Federal Catarinense – Campus Rio do Sul
wiliamwessner123@gmail.com, rodrigo.curvello@ifcc-riodosul.edu.br

Abstract. *The process of teaching Calculus has been a challenge for teachers. This paper describes the development of the Derivatives Game, which is a memory game to stimulate students of Calculus to memorize as trivial derivatives. In the implementation of the game were used 4 threads and a programming language chosen for Java.*

Key-words: *Derivative; Memory game; Parallel Programming.*

Resumo. *O processo de ensino aprendizagem de Cálculo tem sido um desafio para os docentes. Este trabalho descreve o desenvolvimento do Jogo das Derivadas, que é um jogo da memória para estimular estudantes de Cálculo a memorizar as derivadas triviais. Na implementação do jogo foram utilizadas 4 threads e a linguagem de programação escolhida foi o Java.*

Palavras-chave: *Derivada; Jogo da Memória; Programação Paralela.*

1. Introdução

O Jogo das Derivadas foi desenvolvido durante a disciplina de Programação Paralela e Multi-core e tem como objetivo auxiliar os discentes de Cálculo I na memorização das principais derivadas.

No presente trabalho são mostradas as principais características do Jogo das Derivadas. A linguagem de programação utilizada foi o Java, e como a maioria dos jogos envolvem programação concorrente e *Threads*, o Jogo das Derivadas não deixa de ser diferente.

2. Derivada

Segundo Anton (2000, p. 179) a derivada f' de uma função pode ser interpretada ou como uma função cujo valor em x é a inclinação da reta tangente ao gráfico $y = f(x)$ em x , ou, alternativamente, como uma função cujo valor em x é a taxa instantânea da variação de y em relação à x .

De acordo com Santana (2010, p. 29) o estudo da derivada apresenta diversas aplicações práticas, ela é constantemente aplicada em muitos problemas que envolvem o dia-a-dia do ser humano, possibilitando até mesmo resolver situações que envolvam taxas de variação. Santana (2010, p. 29) salienta que o cálculo de derivadas que tem

importância especial em virtude das inúmeras aplicações em vários campos das ciências, tais como: problemas da física, biologia, química, modelagem matemática, arquitetura, geologia, engenharia e economia.

3. Thread

Uma *thread*, como definido Cordeiro (2004) é um artifício que permite a coexistência de múltiplas atividades dentro de um único processo. Java é a primeira linguagem de programação a incluir explicitamente o conceito de *threads* na própria linguagem. *Threads* são também chamados de “processos leves”, pois, da mesma forma que processos, threads são independentes, possuem sua própria pilha de execução, seu próprio *program counter* e suas próprias variáveis locais. Porém, threads de um mesmo processo compartilham memória, descritores de arquivos (*file handles*) e outros atributos que são específicos daquele processo.

Um problema com aplicativos de uma única *thread* é que pode levar a uma fraca responsividade é que as atividades longas e demoradas devem ser concluídas antes de as outras poderem iniciar (DEITEL, H. M., 2010; DEITEL, P. J., 2010).

O Jogo das Derivadas utiliza 4 *threads* na sua execução:

- 1 *thread* para executar a música de fundo.
- 1 *thread* para contar o tempo de jogo.
- 1 *thread* para desvirar as cartas.
- 1 *thread main* do jogo.

4. Processo de ensino aprendizagem de Cálculo

O ensino de Cálculo Diferencial e Integral está cada vez mais presente nos Cursos Superiores, e mostra a importância da disciplina para o desenvolvimento do conhecimento científico, mas o desempenho insatisfatório dos alunos nessa disciplina tem preocupado pesquisadores de todo o mundo, com níveis altíssimos de reprovação e desistências em cursos de Licenciaturas e Engenharias, conforme salienta Silva (2009).

Segundo a pesquisa realizada por Gregor et al. (s.d, p. 12) 55,2% dos alunos avaliam a disciplina de Cálculo como difícil e 23,8 % como muito difícil. Pensando nisso, foi desenvolvido o Jogo das Derivadas, que procura familiarizar os estudantes com as principais derivadas.

5. Jogo das Derivadas

O Jogo das Derivadas é um jogo da memória um pouco diferente dos tradicionais, no lugar de pares iguais, as cartas são compostas por uma função $f(x)$ ou $f'(x)$. O objetivo do jogo é que dado uma carta $f(x)$ deve-se encontrar a outra carta que represente $f'(x)$, uma imagem do jogo pode ser visto na figura 1.

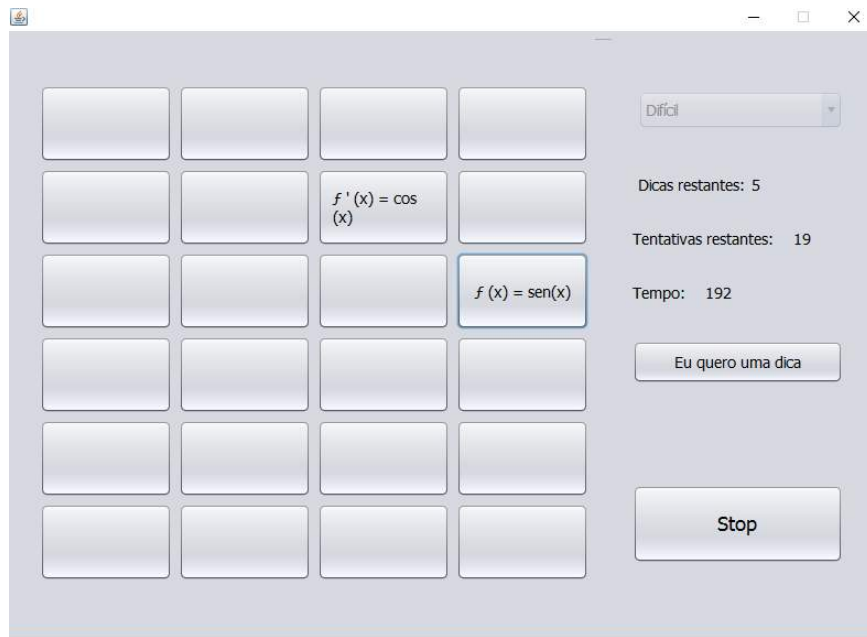


Figura 1: Jogo das Derivadas

O Jogo das Derivadas foi desenvolvido em Java, portanto, pode ser utilizado em qualquer sistema operacional que tenha o JRE (*Java Runtime Environment*) instalado.

5.1. Pares de cartas do jogo

O Jogo das Derivadas é composto por 20 pares de cartas, onde a primeira carta do par representa uma $f(x)$ e a segunda carta $f'(x)$, conforme pode ser visto na tabela 1.

Primeira carta do par	Segunda carta do par
$f(x) = 7x$	$f'(x) = 7$
$f(x) = 3x^2$	$f'(x) = 6x$
$f(x) = x^4$	$f'(x) = 4x^3$
$f(x) = \prod$	$f'(x) = 0$
$f(x) = e^x$	$f'(x) = e^x$
$f(x) = \text{sen}(x)$	$f'(x) = \cos(x)$
$f(x) = \cos(x)$	$f'(x) = -\text{sen}(x)$
$f(x) = \text{tg}(x)$	$f'(x) = \sec^2(x)$
$f(x) = \sec(x)$	$f'(x) = \sec(x) * \text{tg}(x)$
$f(x) = \text{cosec}(x)$	$f'(x) = \text{cosec}(x) * \text{cotg}(x)$
$f(x) = \text{cotg}(x)$	$f'(x) = \text{cosec}^2(x)$
$f(x) = \ln(x)$	$f'(x) = 1/x$
$f(x) = x$	$f'(x) = 1$

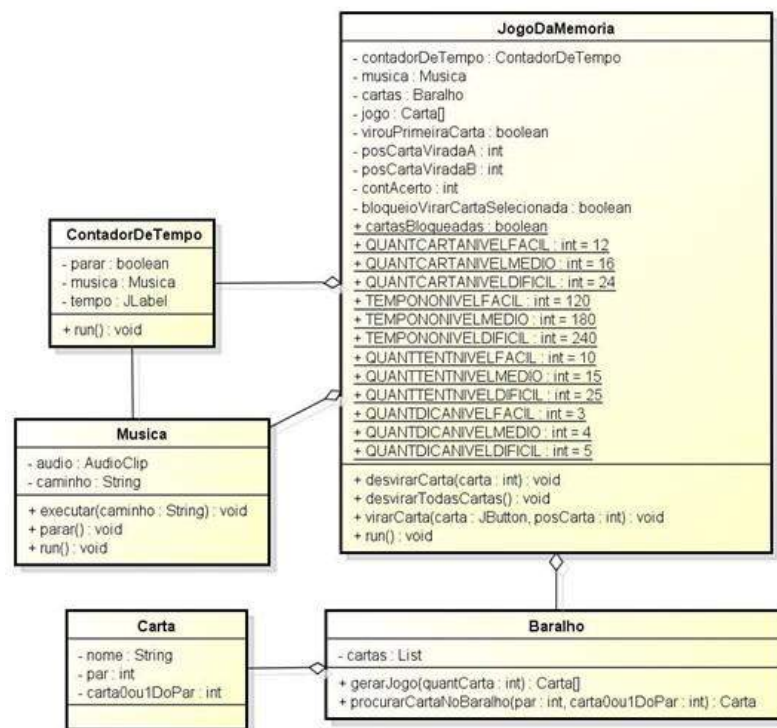
$f(x) = x^2$	$f'(x) = 2x$
$f(x) = 5x$	$f'(x) = 5$
$f(x) = 2x$	$f'(x) = 2$
$f(x) = 200x$	$f'(x) = 200$
$f(x) = 500x$	$f'(x) = 500$
$f(x) = 30x^2$	$f'(x) = 60x$
$f(x) = 8x$	$f'(x) = 8$

Tabela 1: Pares de cartas do Jogo das Derivadas

O Jogo das Derivadas tem 3 níveis de dificuldade. O jogo tem o nível fácil, médio e difícil, com respectivamente, 6, 8 e 12 pares de cartas. Para que o jogo fique dinâmico, os pares de cartas são sorteados pelo método gerarJogo da classe Carta, conforme o Diagrama de Classe da figura 2.

5.2. Diagrama de Classe

Segundo Booch (2012, p. 117), “um diagrama de classe é um conjunto de classes, interfaces e colaborações e seus relacionamentos”.

**Figura 2 – Diagrama de classes do Jogo das Derivadas**

O diagrama de classes, mostrado na Figura 2, representa as classes implementadas e as relações entre elas do jogo.

5.2.1 Classe Contador de Tempo

A classe Contador de Tempo é responsável por fazer a contagem regressiva de tempo, sendo que em cada nível é dado um tempo para o usuário vencer o jogo. Foi utilizado uma thread para cuidar da contagem de tempo. O código pode ser visto na figura 3.

```
public void run() {
    int cont = Integer.parseInt(tempo.getText());
    while (!parar) {
        try {
            tempo.setText("" + cont);
            Thread.sleep(1000);
            cont--;
            if(cont == 0){
                musica.parar();
                musica.executar("/musicas/fimdotempo");
                JogoDaMemoria.cartasBloqueadas = true;
                JOptionPane.showMessageDialog(null, "Ahh! O seu tempo "
                    + "acabou. Não desista e tente novamente.");
                setParar(true);
            }
        } catch (InterruptedException ex) {
            Logger.getLogger(ContadorDeTempo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figura 3 – Método *run* da classe Contador de Tempo

Quando o tempo de jogo acaba, ou seja, chega a zero, o usuário recebe uma mensagem avisando e é executado um som de que o tempo acabou.

5.2.2 Classe Música

A classe Música implementa a interface *Runnable*, o método *run* é responsável por executar a música, conforme a figura 4.

```
public class Musica implements Runnable {
    private static AudioClip audio; //musica pra tocar
    private String caminho; //caminho da musica

    @Override
    public void run() {
        executar(caminho);
    }

    public void parar() {
        audio.stop();
    }

    public void executar(String caminho) {
        URL url = Musica.class.getResource(caminho+".wav");
        audio = Applet.newAudioClip(url);
        audio.play();
    }
}
```

Figura 4 – Classe Música

O Jogo das Derivadas tem 4 sons no pacote músicas que podem ser executadas no decorrer do jogo. Existe a música principal do jogo, que é executada quando o usuário está jogando. O jogo tem também um áudio quando o jogador vence e outros 2 áudios quando perde jogo.

5.2.3 Método *run* da classe Jogo da Memória

A classe Jogo da Memória também implementa a interface *Runnable*, onde o método *run* é responsável por mostrar as duas cartas por 2 ms e depois desvirá-las. O código pode ser observado na figura 5.

```
@Override
public synchronized void run() {
    bloqueioVirarCartaSelecionada = true;
    try {
        Thread.sleep(2000);
    } catch (InterruptedException ex) {
        Logger.getLogger(JogoDaMemoria.class.getName()).log(Level.SEVERE, null, ex);
    }
    desvirarCarta(posCartaViradaA);
    desvirarCarta(posCartaViradaB);
    bloqueioVirarCartaSelecionada = false;
}
```

Figura 5 – Método *run* da classe Jogo da Memória

A utilização de uma *thread* para fazer este efeito de virar e desvirar as cartas é necessária para evitar que o jogo trave durante a execução.

6. Conclusão

Neste artigo foi apresentado o Jogo das Derivadas, foi possível abordar aspectos importantes sobre seu desenvolvimento, requisitos para utilização, seu funcionamento, etc.

A Programação Paralela tem sido utilizada em diversos jogos, no Jogo das Derivadas não foi diferente. Como o jogo tem uma música de fundo, um contador de tempo e o efeito de virar e desvirar as cartas, foi necessário utilizar *threads* para executarem paralelamente.

Referências

- ANTON, Howard. **Cálculo, um novo horizonte**. Porto Alegre: Bookman, 2000. 578 p.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário**. Tradução de: Fábio Freitas da Silva e Cristina de Amorim Machado. Rio de Janeiro.
- CORDEIRO, Daniel de Angelis. **Introdução ao uso de Threads em Java**. 2004. 12 p. - USP, [S.l.], 2004. Disponível em: <<https://www.ime.usp.br/~gold/cursos/2004/mac438/threadsEmJava.pdf>>. Acesso em: 19 mar. 2017.
- DEITEL, Harey M.; DEITEL, Paul J. **JAVA: Como Programar**. 8ª Edição, Prentice Hall, 2010.

GREGOR, Isabela Cristina Soares et al. **Dificuldades de ensino e aprendizagem de cálculo diferencial e integral no IFNMG e o uso de softwares matemáticos**. 15 p. Instituto Federal de Educação, Ciência e Tecnologia do Norte de Minas Gerais (IFNMG) - Campus Januária, [s.d.].

Disponível em: < http://www.senept.cefetmg.br/galerias/Anais_2014/GT01/GT_01_x7x.pdf >.

Acesso em: 18 mar. 2017.

SANTANA, Anderson Marcolino de. **Aplicação das Derivadas**. 2010. 48 p. Trabalho de Conclusão de Curso (Curso de Matemática) - Universidade Federal de Rondônia, Ji-paraná, 2010. Disponível

em: <http://www.dmejp.unir.br/menus_arquivos/1787_anderso_marcolino.pdf>. Acesso em: 14 maio 2017.

SILVA, B. A. **Componentes do processo de ensino e aprendizagem de cálculo: saber, aluno e professor**. In: Seminário Internacional de Pesquisa em Educação Matemática, 4, 2009, Brasília.

Anais... Brasília: Sociedade Brasileira de Educação Matemática, 2009.

UTILIZAÇÃO DOS SISTEMAS DE SINGLE SIGN-ON (SSO) DO GOOGLE E FACEBOOK

Marcos Momm¹, Rodrigo de Moraes¹, Wiliam Wessner¹, Wesley dos Reis Bezerra²

¹ Aluno da sétima fase do curso de Bacharel em Ciência da Computação do Instituto
Federal Catarinense, Campus Rio do Sul

² Professor do Instituto Federal Catarinense, Campus Rio do Sul
marcos.momm@hotmail.com, MoraesRodrigo6@gmail.com,
wiliamwessner123@gmail.com; wesley@ifc-riodosul.edu.br

Abstract. *This article presents a brief definition of login management system, Single Sign-On, as well as an explanation of the OAuth protocol, normally used by this systems. Besides this, will be done a comparison between Google's and Facebook's login management systems, in order to demonstrate the implementation of two distincts Single Sign-On Systems and verificate their difference and similarity.*

Keywords: *Login Management System; Single Sign-On; OAuth;*

Resumo. Este artigo apresenta uma breve definição de sistemas de gerenciamento de login, *Single Sign-On* (SSO), bem como uma explanação do protocolo *OAuth*, normalmente utilizado por estes sistemas. Além disto, estará sendo feita uma comparação entre os sistemas de autenticação do Google e Facebook, a fim de demonstrar as implementações de dois sistemas de *Single Sign-On* diferentes e verificar suas diferenças e semelhanças.

Palavras-chave: *Sistema de Gerenciamento de Autenticação; Single Sign-On; OAuth;*

1. Introdução

É muito comum nos dias de hoje, com a popularização da internet, a utilização de *sites* e diversos serviços *on-line*, desde redes sociais e sites de entretenimento à *sites* de compra (*e-commerces*) e sistemas bancários. Deste modo, para que o usuário possa prover destes serviços, normalmente é necessário realizar o seu cadastramento na página *web* desejada para poder utilizá-la e ser identificado pelo sistema. Porém isto pode ser um certo problema para alguns usuários, pois, com o volume de *websites* utilizados, aumenta o número de credenciais de acesso diferentes que o usuário possuirá para realizar as autenticações em seus serviços, o que poderá vir a gerar problemas com a administração de senhas e *login* em sistemas heterogêneos.

Visando resolver este problema, existem soluções de *Single Sign-On* (SSO) que tem por objetivo proporcionar ao usuário a possibilidade de autenticar-se em múltiplos serviços a partir da autenticação em apenas uma instância de identificação. Para gerir os *logins* através de uma solução *Single Sign-On* é utilizado o protocolo de autenticação *OAuth*. De acordo com Mafrá (2010), o principal objetivo do *OAuth* é permitir que uma aplicação se autentique em outra “em nome de um usuário”, sem precisar ter acesso a senha dele. Um exemplo clássico disto é quando, em alguns serviços *on-line*, há a possibilidade de realizar a autenticação utilizando contas de redes sociais, tais como Google, Facebook ou até mesmo Twitter.

Normalmente algumas páginas web oferecem a possibilidade do usuário realizar o cadastro criando uma nova credencial, ou também, há a opção de vincular esta nova conta, com as contas do Google ou Facebook, onde caso o usuário escolha esta opção, o *website* em que o usuário está se autenticando pedirá uma autorização para realizar tal vínculo.

Neste trabalho, além de explicar o funcionamento das soluções de *Single Sign-On*, estará sendo feita uma breve comparação entre os sistemas de login do Google e do Facebook, utilizando a linguagem de programação PHP, a fim de demonstrar e comparar ambas as implementações, baseadas numa solução de *Single Sign-On* utilizando o protocolo *OAuth*.

2. Single Sign On (SSO)

Single Sign-On (SSO) é um mecanismo pelo qual torna-se possível que um usuário obtenha acesso a múltiplos serviços após autenticar somente uma vez em qualquer um destes serviços. Um exemplo é os serviços do Google, onde, tudo está centralizado nas contas do Google em que somente com um login e senha o usuário pode utilizar todos os serviços que o Google oferece, como o *Gmail*, *YouTube* e demais plataformas do Google, (PASSAPORTEWEB, 2011). Conforme figura 1 que demonstra a utilização do SSO. O usuário pode logar somente uma vez e usar os serviços daquele website em que o usuário fez o login.

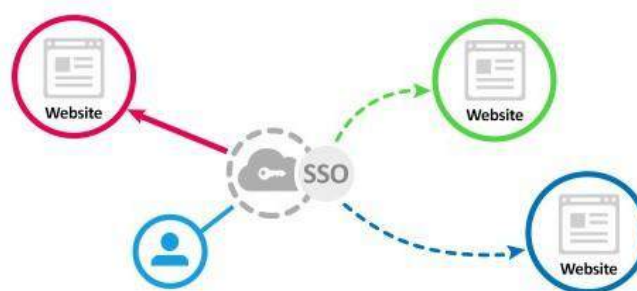


Figura 1: Acesso a diversos websites utilizando o SSO

Fonte: <https://www.oneall.com/services/single-sign-on/>, acessado em maio/2017.

Para o funcionamento do sistema SSO é utilizado o protocolo *OAuth*, a utilização deste protocolo possibilita a comunicação entre os servidores da aplicação e autenticação de forma controlável, utilizando mecanismos de controle de acesso preexistentes, utilizados nos mais diversos contextos.

2.1 Autenticação e Autorização

Autenticação é o processo de verificação de uma identidade alegada, por meio de comparação das credenciais apresentadas pelo cliente com outras pré-definidas. A combinação *username/password* é muito comum mas vários outros métodos estão disponíveis, como os certificados digitais, biometria e dentre outros meios de autenticação (MORAES, 2013).

Autorização é o processo que ocorre após a autenticação e que a função de diferenciar os privilégios atribuídos ao cliente autenticado. Os atributos de autorização normalmente são definidos em grupos mantidos em uma base de dados centralizados (MORAES, 2013).

2.2 Protocolo OAuth

O protocolo *OAuth* é permitir que uma aplicação se autentique em outra sem precisar ter acesso à senha do usuário. A aplicação pede permissão de acesso ao usuário e com isso o usuário concede ou não permissão a determinada aplicação sem a necessidade de digitar a senha ou informar dados pessoais. O processo de autenticação por meio do protocolo *OAuth* é realizado em três passos: aplicação cliente obtém chave de autenticação, usuário autoriza aplicação cliente na aplicação servidora e o último passo a aplicação cliente troca a chave de autenticação pela chave de acesso, (MAFRA, 2010).

No primeiro passo, a aplicação cliente obtém uma chave de autenticação, onde, o cliente solicita à aplicação servidora uma chave de autenticação. O servidor devolve duas chaves: uma pública e uma privada; essas chaves são utilizadas no processo de autenticação e não dão permissão de acesso.

No segundo passo o usuário autoriza aplicação cliente na aplicação servidora. A aplicação servidora redireciona o usuário de volta à aplicação cliente e o cliente obtém uma chave de acesso usando a chave privada e a aplicação servidora devolve apenas uma chave de acesso que apenas a aplicação cliente conhece, todo esse fluxo é representado na figura 2. Em que o usuário é representado pelo *notebook* e a aplicação cliente e a aplicação servidora são as CPU's.

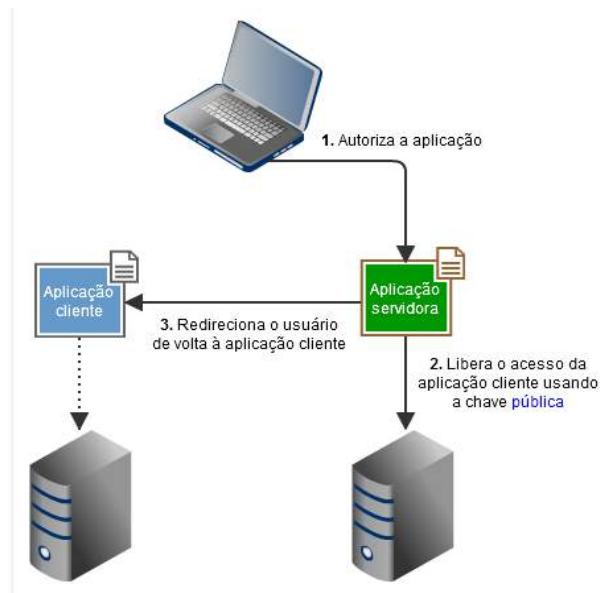


Figura 2: Processo de autorização da aplicação cliente na aplicação servidora

Fonte: <http://www.diogomafra.com.br/2010/09/como-funciona-autenticacao-oauth.html>, acessado em Maio/2017.

No terceiro passo a aplicação cliente troca a chave de autenticação pela chave de acesso, a chave de acesso só é concedida se o usuário permitir o acesso da aplicação. Para fazer a comunicação com a aplicação servidora é utilizada a chave privada, desse modo, apenas a aplicação cliente consegue obter a chave de acesso que é utilizada pelo cliente para acessar a API do servidor “em nome” do usuário. Todo esse processo é demonstrado conforme a figura 3.

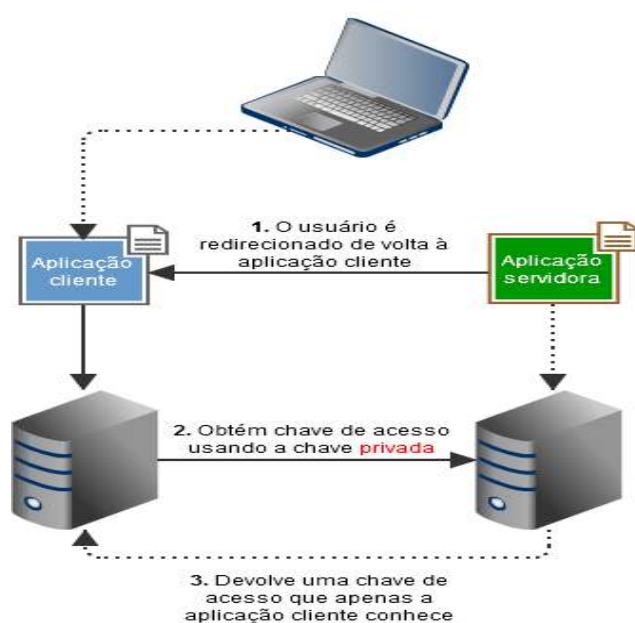


Figura 3: Aplicação cliente trocando a chave de autenticação pela chave de acesso

Fonte: <http://www.diogomafra.com.br/2010/09/como-funciona-autenticacao-oauth.html>, acessado em Maio/2017.

3. Métodos e ferramentas utilizados

Para implementação dos sistemas de autenticação do Google e do Facebook, neste exemplo, devido a sua simplicidade e sua apropriação ao desenvolvimento *web*, foi usada a linguagem de programação PHP em sua versão 5.5.8, juntamente com o servidor Apache, na versão 2.0. Além disto, como IDE de desenvolvimento foi utilizado o NetBeans na versão 8.0.2.

4. Implementação do sistema de autenticação do Facebook e do Google

De acordo com Miculan e Urban (2011) uma chamada de método do Facebook é feita através da Internet enviando pedidos HTTP GET ou POST para o servidor Facebook REST da API.

Para implementar o sistema de autenticação do Facebook, foi necessário criar um aplicativo no Facebook. Com o aplicativo criado, basta criar uma âncora no código HTML e passar os dados *client_id*, *redirect_uri* e *scope* pela URL, conforme a figura 3.

```
<a href="https://www.facebook.com/dialog/oauth?
  client_id=710954592410644&
  redirect_uri=http://localhost/loginFacebook/index.php/&
  scope=email,user_website,user_location">Entrar com Facebook
</a>
```

Figura 3: URL preenchida com os dados do App

Fonte: Elaborado pelos autores, 2017.

Os parâmetros obrigatórios são o *client_id* e *redirect_uri*, e devem ser preenchidos de acordo com o aplicativo no Facebook. O *scope* é opcional, e por padrão o Facebook fornece id do usuário, nome, sexo e localidade. Na figura 3 foi solicitado *e-mail*, *user_website* e *user_location* adicionais. A tabela 1 apresenta uma descrição dos parâmetros que são enviados via URL.

Parâmetros	Descrição
<i>client_id</i>	É um identificador da aplicação de autenticação que serve para identificar a aplicação.
<i>redirect_uri</i>	É uma URL de retorno, ou seja, para onde a página vai ser redimensionada depois do <i>click</i> .
<i>scope</i>	É inserida as permissões adicionais que o <i>site</i> precisa.

Tabela 1: Parâmetros enviados pela URL

Fonte: Adaptado de Facebook, 2017.

Quando o usuário clicar no *link* será redirecionado para fazer autenticação, depois o Facebook pergunta se deseja compartilhar as informações com o *site*. Quando aceito compartilhar as informações com o *site*, basta gravar as informações em um repositório, para que possa ser utilizado posteriormente.

A figura 4 apresenta um trecho da implementação do sistema de autenticação. Neste código, basicamente, é iniciada uma aplicação com os dados, obtida uma sessão e faz a requisição dos dados solicitados.

```
//Inicia a aplicação com id, chave secreta e redirect
FacebookSession::setDefaultApplication('710954592410644',
    '56b40a1e488bfb0e25dadedc5314cdaf');
$redirect = new FacebookRedirectLoginHelper(
    'http://localhost/loginFacebook/1353/fbconfig.php');

$sessao = $redirect->getSessionFromRedirect();

//verifica se a sessão existe
if (isset($sessao)) {
    // graph api para requisitar os dados do usuário
    $request = new FacebookRequest($sessao, 'GET', '/me');
    $resposta = $request->execute();
    $graphObject = $resposta->getGraphObject();

    //Get ID do usuário do Facebook
    $id = $graphObject->getProperty('id');
    //Get nome completo
    $nome = $graphObject->getProperty('name');
    //Get email
    $email = $graphObject->getProperty('email');
}
```

Figura 4: Requisição dos dados
Fonte: Elaborado pelos autores, 2017.

O sistema de autenticação do Google é bastante semelhante com o do Facebook, sendo necessário criar uma conta de desenvolvedor no Google para que possa ser gerado o id do cliente, chave secreta e a URLs de redirecionamento autorizados.

A utilização do SSO permite ao usuário ter todos os serviços centralizados em uma única conta, e com isso o usuário não precisa estar decorando várias senhas, para cada serviço que o usuário vai utilizar.

Um dos perigos do protocolo *OAuth* é que como o usuário não precisa fornecer a senha, o usuário se sente seguro. As autorizações devem ser dadas a apenas aplicações que o usuário confia, pois, com essa autorização, uma aplicação mal intencionada pode trazer um certo prejuízo ao usuário, um detalhe importante é revogar a autorização de aplicações que não são mais utilizadas ou são de origem duvidosa (MAFRA, 2010).

5. Conclusão

Com o aumento crescente das Redes Sociais, em muitos sites em que precisa ser feito um login, é mais cabível utilizar a autenticação do Google ou do Facebook, poupando assim o usuário de ter que preencher todo um cadastro de acesso. Com o uso das chaves de acesso o usuário pode ter todos os serviços centralizados, acompanhando tudo o que acontece.

A chave de autenticação do Facebook e do Google funcionam de forma semelhante, onde são requeridas as informações do usuário, caso o usuário aceite, estas informações podem ser utilizadas na aplicação.

Uma das vantagens da utilização do SSO e do protocolo OAuth é que o usuário consegue ter um único login e ter acesso a mais diversas aplicações. Um bom exemplo é o Google em que centraliza todos os serviços nas contas Google, ou seja, o usuário loga apenas uma vez e tem acesso a todos os serviços.

A utilização deste tipo de autenticação tem como principal desvantagem a questão de segurança. Quando um usuário tem uma conta de rede social invadida, o invasor terá acesso a todas as outras aplicações que estão conectadas.

Referências

FACEBOOK. **Criar manualmente um fluxo de login**. s.d.. Disponível

em:<<https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow/>>. Acesso em: 06 junho 2017.

MAFRA, Diogo Edegar. **Como funciona a autenticação OAuth**. 2010. Disponível

em:<<http://www.diogomafra.com.br/2010/09/como-funciona-autenticacao-oauth.html>>. Acesso em: 11 maio 2017.

MICULAN, Marino; URBAN, Caterina. **Formal analysis of Facebook Connect single sign-on authentication protocol**. In: SOFSEM. 2011. p. 22-28. Disponível em:

[http://s3.amazonaws.com/academia.edu.documents/42071177/Formal_analysis_of_Facebook_Connect_Sing20160204-32378-j8zoiq.pdf?](http://s3.amazonaws.com/academia.edu.documents/42071177/Formal_analysis_of_Facebook_Connect_Sing20160204-32378-j8zoiq.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1496783158&Signature=UwnzDYX1KL%2F%2B%2BAJLd3WAQ4%2FF7oM%3D&response-content-disposition=inline%3B%20filename%3DFormal_analysis_of_Facebook_Connect_Sing.pdf)

[AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1496783158&Signature=UwnzDYX1KL%2F%2B%2BAJLd3WAQ4%2FF7oM%3D&response-content-disposition=inline%3B%20filename%3DFormal_analysis_of_Facebook_Connect_Sing.pdf](http://s3.amazonaws.com/academia.edu.documents/42071177/Formal_analysis_of_Facebook_Connect_Sing20160204-32378-j8zoiq.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1496783158&Signature=UwnzDYX1KL%2F%2B%2BAJLd3WAQ4%2FF7oM%3D&response-content-disposition=inline%3B%20filename%3DFormal_analysis_of_Facebook_Connect_Sing.pdf) > Acesso em: 06 junho 2017.

MORAES, Alexandre M. S. P. **Autenticação, Autorização e Accounting: Conceitos Fundamentais**. 2013. Disponível em:

<<https://alexandremspmoraes.wordpress.com/2013/02/15/autenticacao-autorizacao-e-accounting-conceitos-fundamentais/>>. Acesso em: 18 maio 2017.

PASSAPORTEWEB. **Single Sign On**. 2011. Disponível

em:<<https://app.passaporteweb.com.br/static/docs/sso.html>>. Acesso em: 11 maio 2017.

COMPARATIVO ENTRE BANCOS DE DADOS NÃO RELACIONAIS – NOSQL

Benny Roger Zimmer¹, Lucas Grigolon Varela¹, Matheus Herique Alves¹, Silvio Regis da Silva Junior¹, Wesley dos Reis Bezerra²

¹Cursando Bacharelado em Ciência da Computação no Instituto Federal Catarinense – Campus Rio do Sul

²Professor do Instituto Federal Catarinense – Campus Rio do Sul

benny274@msn.com, lucasgvarela@gmail.com, mateush.alves96@gmail.com,
silvio.regis@outlook.com, wesley.bezerra@ifc.edu.br

Abstract. *Currently the academic society initiates the studies in SQL and DBs (Data Base) using DBMS (Database Management Systems). However there are several other types of databases currently, which may even perform better for certain activities. A DB model that has been increasingly used by companies with a large volume of data is NoSQL. In this paper we'll see a little more about what this type of DB is and a brief comparison between the main DBs that use this model, The Couch DB, Mongo DB and Cassandra.*

Key-words: *Database; NoSql; Non Relational.*

Resumo. *Atualmente a sociedade acadêmica inicia os estudos em SQL e BDs (Bancos de Dados) utilizando SGBDs (Sistemas Gerenciadores de Banco de Dados). Porém existem vários outros tipos de bancos de dados atualmente, que podem até apresentar um melhor desempenho para determinadas atividades. Um modelo de BD que atualmente tem sido cada vez mais utilizado por empresas que possuem um grande volume de dados é o NoSQL. Neste artigo veremos um pouco mais sobre o que é esse tipo de BD e uma breve comparação entre os principais BDs que utilizam este modelo, O Couch DB, Mongo DB e Cassandra.*

Palavras-chave: *Banco de dados; NoSql; Não-Relacional.*

1. Introdução

A maioria dos bancos NoSQL são Open Source e surgiram para trazer mais eficiência e robustez nas consultas, permitindo também a escalabilidade no armazenamento e processamento de grandes volumes de dados. Abordaremos neste trabalho 3 bancos de dados não-relacionais: MongoDB, Apache CouchDB e Apache Cassandra.

A maioria das empresas e programadores conhecem bancos de dados relacionais, e utilizam eles na maioria das vezes em seus projetos, muitas vezes sem medir a eficácia entre bancos de dados para obter maior velocidade de entrega das consultas em seus

bancos de dados. Como existem bancos de dados relacionais, existem também, a possibilidade de utilizar bancos de dados não relacionais, também conhecidos por NOSQL.

2. Banco de dados relacionais

Banco de dados relacionais podem ser visto como uma tabela, onde as informações são acessadas através de seus atributos, e essa tabela, pode ser repetidas por definas vezes. Mas ganhou esse nome pelo simples fato, de que, os dados são armazenados de tal forma, que podem existir relações entre as tabelas nele existentes.

Os SGBDs relacionais oferecem aos usuários processos de validação, verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas, segurança, controle de transações, otimização de consultas, dentre outros. A utilização de tais recursos facilitou a vida dos desenvolvedores de aplicações, possibilitando que estes pudessem se preocupar exclusivamente com o foco da aplicação.

Como um dos conceitos mais básicos do modelo relacional, as chaves representam uma forma simples e eficaz de associação entre as tabelas do banco de dados. A chave primária foi criada com o objetivo de identificar de forma única as tuplas da tabela e ainda de determinar a ordem física dessas tuplas. A chave estrangeira permite uma relação de dependência entre atributos de tabelas distintas, de forma que os valores permitidos em um atributo dependem dos valores existentes em outro atributo (BRITO, 2010).

Uma das principais vantagens – talvez seja o motivo maior para sua popularidade – é que além de conceitos o modelo ainda conta com uma técnica de diagramação. Isto permite registrar e comunicar de forma simplificada os principais aspectos do projeto de banco de dados DATE (2004, p. 358).

3. Banco de dados não relacionais

Segundo Diana e Gerosa (2010, página 2) assim como o termo não-relacional, o termo NOSQL não ajuda a definir o que esses bancos são de fato. Além do problema da falta de precisão, esse termo também tem contribuído para uma grande confusão em torno dessa categoria de bancos de dados, já que a princípio a linguagem SQL não é sinônimo de bancos de dados relacionais, nem representa as limitações desses bancos de dados. Devido a isso, o termo NOSQL tem sido usado com o significado de “Não apenas SQL” numa tentativa da comunidade de reconhecer a utilidade dos modelos tradicionais e não divergir as discussões. No fim, NOSQL não define precisamente esses bancos de dados, mas no geral cada um deles apresenta a maioria das seguintes características: não-relacional, distribuído, de código aberto e escalável horizontalmente, ausência de esquema ou esquema flexível, suporte à replicação nativo e acesso via APIs simples (NOSQL, 2010).

Estes modelos de bancos de dados, não possuem tabelas que se tenham relacionamento de cardinalidade com outras tabelas, mas sim coleções de dados com atributos dinâmicos, onde a consulta e a inserção de resultados pode se tornar mais simples e eficiente para um número grande de registros que podem não possuir os mesmos atributos.

MongoDB: O MongoDB combina a capacidade de escalabilidade com recursos como índices secundários, consultas de intervalo, classificação, agregações e índices geoespaciais (CHODOROW, 2013).

Apache CouchDB: é um banco de dados orientado a documento NOSQL que foca na facilidade de uso e em ter uma arquitetura que “abraça completamente a Web”. A arquitetura do banco de dados foi implementada com programação concorrente na linguagem Erlang, usa JSON para armazenamento dos dados, e JavaScript como linguagem de consulta, permite o uso de MapReduce e HTTP para utilizar API's (FOUNDATION, 2017), mantido pela Apache Software Foundation.

Apache Cassandra: Banco de dados distribuído altamente escalável e de alto desempenho projetado para lidar com grandes quantidades de dados em muitos servidores de commodities (FOUNDATION, 2016), mantido pela Apache Software Foundation.

4. Comparativo entre os bancos de dados não relacionais

Para poder fazer um comparativo, utilizamos dados pesquisados e retirados diretamente do site oficial dos desenvolvedores das ferramentas SGBD (Sistemas Gerenciadores de Banco de Dados), o apêndice A exemplifica em forma de tabela as características descritas abaixo:

- É característica comum aos três bancos de dados não relacionais estudados: Suporte a compreensão de dados; atualização de condições de entrada; suporte unicode; permitem *MapReduce*; tem consistência no armazenamento de dados; facilitam a persistência de dados; possuem índices secundários; suportam chaves compostas; são escaláveis; permitem replicação; tem o modo de replicação: *Master-Slave*; permitem *sharding*, tem arquitetura compartilhada e são multiplataforma.
- Características comum ao MongoDB e o CassandraDB: modelo de integridade: BASE.
- Características comum ao MongoDB e o CouchDB: armazenamento em sistema de arquivo de memória volátil.
- Características comum ao CassandraDB e o CouchDB: permitem atomicidade (conceito de transação).
- O MongoDB tem: Índices geoespaciais, atomicidade de tipo condicional, feito em C++.
- O CassandraDB tem como particularidade: linguagem de consulta: Cassandra Query Language (CQL), feito em Java.
- O CouchDB tem: Modelo de integridade: MVCC. Permite isolamento, permite controle de revisão, feito em Erlang, C++, C e Python.
- Nenhum deles tem: Integridade referencial; suporte a transações; pesquisa completa de texto ou suporte a gráficos.

5. Conclusão

Cada um dos bancos de dados que foram comparados tem suas particularidades, o que faz com que seja possível utilizar cada um com uma aplicação diferente. Para que fosse possível realizar uma comparação justa entre cada um, foram mostradas o que cada um possui e difere do outro.

Se for feita uma comparação entre as funcionalidades de cada um, o Apache Cassandra pode ser considerado o melhor banco de dados não relacional, porém se você precisar de consultas dinâmicas ou se você preferir definir os índices em vez de mapear para reduzir funções a escolha será o MongoDB, pois além de ter estas funções ele é um banco de dados ao qual possui um ótimo desempenho.

Se o mais importante na aplicação é acumular dados e ocasionalmente alterar dados, o melhor banco de dados para isso é o Apache CouchDB nos quais consultas predefinidas devem ser executadas para ganho de desempenho. Apache CouchDB é mais utilizado em locais onde o controle de versão é importante.

Cada banco de dados dispõe de características distintas, elas podem ser úteis dependendo da utilidade em que o banco de dados vai ter na sua aplicação.

Referências

CHODOROW, K. **MongoDB: The Definitive Guide**. 2. ed.: O'reilly Media, Inc., 2013. 432 p.

Disponível em: <https://books.google.com.br/books?id=pAbSHFi4WSAC>. Acesso em: 11 de maio de 2017.

DIANA, M.; GEROSA, M. A. **NOSQL na Web 2.0: Um estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0**. IX Workshop de Teses e Dissertações em Banco de dados, 2010. Disponível em:

http://www.lbd.dcc.ufmg.br/colecoes/wtdbd/2010/sbbd_wtd_12.pdf. Acesso em 13 de maio de 2017.

ENTERPRISE, D. **NoSQL Performance Benchmarks: Cassandra vs. MongoDB vs. Couchbase vs. HBase**. Disponível em: <http://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-hbase>. Acesso em: 11 de maio de 2017.

FOUNDATION, A. S. **About**. Disponível em: <http://couchdb.apache.org/>. 2017. Acesso em: 12 de maio de 2017.

FOUNDATION, A. S. **What is Cassandra?** Disponível em: <http://cassandra.apache.org/>. 2016. Acesso em: 12 de maio de 2017.

MONIRUZZAMAN, A. B. M.; HOSSAIN, S. A. **NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison**. 6. ed.: International Journal Of Database Theory And Application, 2013.

NOSQL. **NOSQL Databases**. Disponível em: <http://nosql-database.org/>. 2010. Acesso em: 11 de maio de 2017.

PANIZ, D. **NoSQL: Como armazenar os dados de uma aplicação moderna**. Casa do Código. 2016. 177 p. Disponível em: <https://books.google.com.br/books?id=Jii5DAAAQBAJ>. Acesso em: 11 de maio de 2017.

BRITO, W. R. **Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa**. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>. Acesso em: 08 de Junho de 2017.

DATE, C. J.. **INTRODUÇÃO A SISTEMAS DE BANCOS DE DADOS**. 8. ed. Rio de Janeiro: Elsevier, 2003.

Apêndices

Características	MongoDB	Apache CouchDB	Apache Cassandra
Data Storage	Volatile memory File System	Volatile memory File System	-
Query Language	Volatile memory File System	JavaScript Memcached - protocol	API calls CQL Thrift
Protocol	Custom, binary (BSON)	HTTP, REST	Thrift & custom binary CQL3
Compression	YES	YES	Yes
Condition entry updates	YES	YES	No
MapReduce	YES	YES	Yes
Unicode	YES	YES	Yes
TTL for Entries	YES	YES	Yes
Integrity model	BASE	MVCC	BASE
Atomicity	Conditional	YES	Yes
Consistency	Yes	YES	Yes
Isolation	No	YES	No
Durability (data storage)	Yes	YES	Yes
Transactions	No	NO	No
Referential integrity	No	NO	No
Revision control	No	YES	No
Secondary Indexes	Yes	YES	Yes
Composite keys	Yes	YES	Yes
Full text search	No	NO	No
Geospatial Indexes	Yes	NO	No
Graph support	No	NO	No
Horizontal scalable	Yes	YES	Yes
Replication	Yes	YES	Yes
Replication mode	Master-Slave-Replication	Master- Slave Replication	Master-Slave Replication
Sharding	Yes	YES	Yes
Shared nothing architecture	Yes	YES	Yes
Value size max.	16MB	20MB	2GB
Operating system	Cross-plataform	Ubuntu, Red Hat, Windows, Mac OS X	Cross-plataform
Programming language	C++	Erlang, C++, C e Python	Java

Apêndice

A – Tabela comparativa entre os bancos de dados não relacional MongoDB, Apache CouchDB e Apache Cassandra.