INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PARÁ

CURSO DE GRADUAÇÃO TECNOLÓGICA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GLAUBER MATTEIS GADELHA

**Gerador de quadros de apresentação de dados em formato de Dashboards**

BELÉM

2020

GLAUBER MATTEIS GADELHA

**Gerador de quadros de apresentação de dados em formato de Dashboards**

Trabalho de Conclusão de Curso apresentado à coordenação do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Pará para obtenção de Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Claudio Roberto de Lima Martins

BELÉM

2020

**RESUMO**

No cenário mundial, a demanda por sistemas digitais cresce constantemente, exigindo alinhamento com as tecnologias mais atuais e uso de ferramentas que acelerem a entrega de aplicações no prazo mais curto possível. No desenvolvimento de softwares, independente da linguagem de programação utilizada, a necessidade de implementação de grandes e complexos blocos de códigos de forma recorrente é presente no dia a dia dos profissionais, portanto, qualquer ferramenta que facilite a aplicação destes blocos padronizados, com as modificações necessárias para o domínio de um problema determinado, pode vir a evitar o desperdício de horas de programação. Em face destas premissas, este trabalho é um estudo de reuso de software e geração de código fonte baseada em gabaritos, com desenvolvimento de uma aplicação que gera um quadro de apresentação de dados em formato gráfico, o que chamamos comumente de Dashboards.

Palavras - chave: Desenvolvimento de aplicações, apresentação de dados, dashboards, gerador de código baseados em gabaritos.

**ABSTRACT**

In the world scenario, the demand for digital systems is constantly growing, requiring alignment with the most current technologies and the use of tools that accelerate the delivery of applications in the shortest possible time. In software development, regardless of the programming language used, the need to implement large and complex code blocks on a recurring basis is present daily in professionals lives, therefore, any tool that facilitates the application of these standardized blocks, with the modifications necessary inside the domain of a specific problem, can avoid wasting hours of programming. In view of these premises, this work is a study of software reuse and template-based source code generation, with the development of an application that generates a data presentation chart in graphic format, which we commonly call Dashboards.

Keywords: Software development, data presentation, dashboards, template-based code gerenation.

Sumário

[1 INTRODUÇÃO 6](#_Toc32043772)

[1.1 Objetivo e delimitação de escopo 7](#_Toc32043773)

[1.2 Procedimentos metodológicos 8](#_Toc32043774)

[1.3 Organização do trabalho 8](#_Toc32043775)

[2 8](#_Toc32043776)

[FUNDAMENTAÇÃO TEÓRICA 8](#_Toc32043777)

[2.1 Reuso de Software 8](#_Toc32043778)

[2.1.2 Técnicas de aplicação de reuso de software 10](#_Toc32043779)

[2.1.3 Geradores de programas 12](#_Toc32043780)

[2.2 Motores de gabarito (*Template Engines*) 13](#_Toc32043781)

[2.2.1 Elementos básicos de motores de gabarito 13](#_Toc32043782)

[2.3.1 Apache Velocity 14](#_Toc32043783)

[2.3.4 Apache Freemarker 14](#_Toc32043784)

[2.3.5 Twig 14](#_Toc32043785)

[3 - Desenvolvimento do Protótipo 15](#_Toc32043786)

[3.1 - Descrição Geral 15](#_Toc32043787)

[3.2 Descrição das Etapas de Desenvolvimento 15](#_Toc32043788)

[3.3 – Levantamento dos Requisitos 16](#_Toc32043789)

[3.4 – Casos de Uso 17](#_Toc32043790)

[5 REFERÊNCIAS 18](#_Toc32043791)

# 1 INTRODUÇÃO

A evolução tecnológica, além de muitas vantagens e confortos, nos trouxe algo que ao mesmo tempo é valioso e assustador: o acúmulo minuto a minuto de massas de dados cada vez maiores. Cientistas da Tecnologia da Informação, Matemáticos e estatísticos vem trabalhando em sofisticadas técnicas voltadas a encontrar padrões nessas massas de dados, transformando caos em informação útil. Ainda assim, os volumes são tamanhos que inviabilizam o entendimento de certos relatórios.

Para minimizar o problema, tem se empregado formas de apresentação de informações sintetizadas em quadros gráficos, chamados Dashboards, que entregam um resumo visual de centenas, muitas vezes milhares de linhas de um conjunto de dados. Isso facilita o entendimento e suporta de forma mais simples e direta a tomada de decisão.

Mesmo com evolução das linguagens de programação além da expansão de suas bibliotecas de Interfaces de Aplicações para Programação, comumente chamadas APIs, ainda é trabalhoso desenvolver a camada de apresentação de dados de forma gráfica e dinâmica. Uma página JSP, ASP ou até mesmo HTML puro, contendo um dashboard que sintetiza em uma página web um conjunto de informações de um relatório de pesquisa em banco de dados requer algumas centenas de linhas de código.

A demanda de mercado por entregas no mais curto prazo possível, bem como a necessidade de se desenvolver artefatos que permitam fácil entendimento, documentação, manutenção e evolução, além da importância de uma apresentação clara e simplificada dos dados, por si só sustentam a necessidade de pesquisa no campo da geração rápida e eficiente de relatórios gráficos. Isto, se feito apropriadamente, reduz a quantidade de falhas de projeto e deixa a equipe de programadores e analistas mais livres para tratar regras de negócio e complexidades típicas de cada domínio de problema. (LUCRÉDIO, 2009).

Poder utilizar-se de bibliotecas ou até mesmo programas standalone que gerem código a partir de um conjunto de definições feitas previamente pelo usuário, o que pode ser chamado de um assistente ou *wizard*, reduz uma etapa trabalhosa e significante do esforço total de desenvolvimento de um sistema completo ou de módulos de um sistema maior. (SHIMABUKURO JUNIOR, 2006). Com um esforço inicial que pode ser razoavelmente grande, pode-se reduzir ou até mesmo eliminar o trabalho posterior em etapas repetitivas e, além disso, manter uma padronização dos artefatos de saída, garantindo a qualidade deste produto final.

Aliando a necessidade de facilitar a exibição e entendimento de informações obtidas a partir de um conjunto de dados ao potencial do reuso de software e da geração de artefatos orientada a gabaritos, a proposta deste trabalho é desenvolver uma aplicação que, recebendo um conjunto de dados em formato aberto, possibilite ao usuário selecionar atributos e gráficos de tipos variados a fim de formar, a partir de um gabarito pré-definido, um Dashboard que apresente de forma clara e com filtragem dinâmica o resumo destes.

## Objetivo e delimitação de escopo

O objetivo deste trabalho é desenvolver uma aplicação em linguagem Java 8 SE para desktop usando técnicas de programação generativa orientada a gabaritos (no inglês, *templates*) que facilite a qualquer usuário comum com um arquivo CSV (Comma Separated Values, ou Valores separados por vírgula) contendo um conjunto de dados, a geração de painéis de informações gerenciais (*Dashboards*) para representação destes dados.

Esta aplicação permitirá ao usuário escolher entre três tipos diferentes de gráficos: gráfico tipo Pizza, gráfico tipo Linha e gráfico tipo Barras Horizontais. Permitirá também dois tipos de reduções: A redução por contagem do total de incidências de um atributo e a redução por somatória, usando um atributo especificado como chave e a somatória de um segundo atributo especificado como valor.

O gabarito trata-se de um arquivo contendo tags HTML e scripts Javascript estáticos, entremeado a marcações da linguagem de template Apache FreeMarker para o conteúdo a ser preenchido em tempo de execução. Os painéis gerados na saída são codificados em HMTL5 com folha de estilos CSS3 e as bibliotecas Javascript DC.js, Crossfilter.js e D3.js.

A interface gráfica de interação usuário / aplicação foi desenvolvida usando JavaFX 8.

## Procedimentos metodológicos

O trabalho foi conduzido em três etapas. No primeiro momento foi feita investigação de literatura e trabalhos acadêmicos a fim fundamentar teoricamente a aplicação, abordando reuso de software, geração automática de código fonte e suas vantagens, aplicação de linguagens de gabaritos, além de técnicas e ferramentas para apresentação de dados em formato dashboard em aplicações web.

Na etapa seguinte, foi feita a análise e modelagem da aplicação, utilizando-se de algumas técnicas já consolidadas de engenharia de software.

Na etapa final, a aplicação foi desenvolvida aplicando as tecnologias e métodos investigados nas etapas anteriores, testada e validada.

## 1.3 Organização do trabalho

Este trabalho de conclusão de curso foi organizado como segue descrito abaixo:

No capítulo 2 é descrita toda a fundamentação teórica e técnica para o trabalho. Compreende uma introdução ao reuso de software, programação generativa e geração de código a partir de linguagens de gabarito, a descrição resumida do histórico e vantagens do motor de gabaritos Apache FreeMarker, e as bibliotecas DC.js, Crossfilter.js e D3.js, responsáveis por apresentar os dados de forma gráfica e dinâmica em páginas HTML5.

No capítulo 3 descreve-se o projeto da aplicação geradora de dashboards e seu desenvolvimento. Por último, no capítulo 4 são feitas as considerações finais.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a base teórica sobre reuso de software, programação generativa, motores de gabaritos e apresentação de dados em formato gráfico dashboard. Além disso, fundamenta tecnicamente as bibliotecas e linguagens utilizadas no desenvolvimento da aplicação final.

## 2.1 Reuso de Software

Uma das definições encontradas na literatura pesquisada com a finalidade de desenvolver o presente trabalho para o reuso de software é o processo de se criar software a partir de software existente, ao invés de simplesmente construí-lo do início (KRUEGER, 1992). Qualquer dos artefatos de software de um projeto anterior e bem sucedido, como código fonte, classes e bibliotecas compiladas, planos, estratégias, diagramas, entre outros, podem e devem ser reutilizados para dar celeridade ao desenvolvimento de outros artefatos ou sistemas. (SHIMABUKURO JUNIOR, 2006).

Lucrédio (2009, p. 30) ressalta em sua tese que reuso de software remonta de 1947, no início da programação armazenada, quando Wheeler e Wilkes desenvolveram o conceito de *jump*, um precursor do comando *goto*, que possibilitava reaproveitar blocos de código dentro do mesmo programa. Desta data adiante, programadores reaproveitam blocos de código em arquivos pessoais, programas antigos, repositórios públicos e até mesmo em sua memória.

A quantidade de software reutilizável disponível tem aumentado significantemente dos anos 2000 até a data atual. O crescimento do movimento de software livre mostra que há uma grande quantidade de soluções disponíveis em repositórios abertos, as quais podemos parametrizar e adaptar aos mais diversos domínios de aplicação a custos acessíveis. Há sistemas completos prontos para serem ajustados às necessidades das mais diversas empresas e a conectividade da rede mundial e serviços web garantem ainda mais opções de reuso (SOMMERVILLE, 2013).

Apesar do potencial naturalmente reusável de componentes e sistemas de software, pode se tornar muito cara sua readaptação para aplicação em um novo domínio. Reaproveitar artefatos devidamente testados e validados certamente pode trazer benefícios na redução do custo total do desenvolvimento de um sistema, entre outras vantagens, porem algumas dificuldades inerentes também a este reaproveitamento podem mostrar que essa redução de custo nem sempre é tão significativa quanto se espera. Há restrições para o reuso influenciadas pela cultura organizacional, orçamento para o projeto, posicionamento pessoal dos indivíduos que compõem o time de análise e desenvolvimento, entre outros (SOMMERVILLE, 2013).

Dentre alguns dos benefícios do reuso de software podemos destacar os seguintes (SOMMERVILLE, 2013):

* Redução do tempo de desenvolvimento: Em muitos projetos, o prazo de entrega da solução acaba por ser mais importante que os custos do desenvolvimento. Neste caso, o reuso se torna praticamente uma necessidade;
* Aumento da confiança: Reusar soluções previamente desenvolvidas, testadas e validadas geralmente reflete uma confiabilidade maior do que uma nova solução desenvolvida;
* Melhor uso da mão-de-obra: Reutilizando software, a equipe de análise e desenvolvimento ganha tempo para dar atenção para novas funcionalidades e regras de negócio.

Quanto aos problemas e dificuldades relacionadas ao reuso, podemos levar em consideração (SOMMERVILLE, 2013):

* Ausência de ferramentas de suporte: Certas ferramentas não dão suporte adequado ao reuso. Isso pode dificultar ou até mesmo impedir o emprego destas ferramentas em um novo sistema;
* Síndrome do “não inventado aqui”: O desenvolvedor pode decidir não usar uma ferramenta pronta pela dificuldade de entendê-la ou por se achar capaz de fazer melhor;
* Aumento do custo de manutenção: Sem acesso ao código fonte, o uso de um componente ou sistema pode se tornar oneroso no tocante à manutenção. Estes componentes podem se tornar incompatíveis ao longo do ciclo de evolução do sistema final.

## 2.1.2 Técnicas de aplicação de reuso de software

Com a evolução do desenvolvimento de software, foram definidas várias técnicas para dar suporte ao reuso. Definir a abordagem mais apropriada para o desenvolvimento de um sistema depende diretamente dos requisitos funcionais e não funcionais, disponibilidade de ativos com possibilidade de reuso e, obviamente, do conhecimento técnico do time de desenvolvimento (SOMMERVILLE, 2013).

A Figura 1 exibe algumas das possibilidades de implementação de reuso de software. Cada uma dessas possibilidades tem descrição resumida na Tabela 1.

Figura 1 – Técnicas para implementação de Reuso de Software



Fonte: Sommerville(2013)

Tabela 1 – Descrição de Abordagens de Reúso de Software.

|  |  |
| --- | --- |
| ABORDAGEM | DESCRIÇÃO |
| Padrões de Arquitetura | Padrões de arquitetura de software que oferecem suporte a tipos comuns de sistemas de aplicação são usados como base de aplicações. |
| Padrões de Projeto | Abstrações genéricas que ocorrem em todas as aplicações são representadas como padrões de projeto, mostrando os objetos abstratos e concretos e as interações. |
| Desenvolvimento Baseado em Componentes | Sistemas são desenvolvidos através da integração de componentes (coleções de objetos) que atendem aos padrões de modelos e componentes. |
| Framework de aplicações | Coleções de classes abstratas e concretas são adaptadas e estendidas para criar sistemas de aplicação. |
| Empacotamento de sistemas legados | Sistemas legados são ‘empacotados’ pela definição de um conjunto de interfaces e acesso a esses sistemas legados por meio dessas interfaces. |
| Sistemas orientados a serviços | Sistemas são desenvolvidos pela ligação de serviços compartilhados, que podem ser fornecidos externamente. |
| Linhas de produtos de software | Um tipo de aplicação é generalizado em torno de uma arquitetura comum para que esta possa ser adaptada para diferentes clientes. |
| Reúso de produtos COTS | Sistemas são desenvolvidos pela configuração e integração de sistemas de aplicação existentes. |
| Sistemas de ERP | Sistemas de grande porte que sintetizam a funcionalidade e as regras de negócios genéricos são configurados para uma organização |
| Aplicações verticais configuráveis | Sistemas genéricos são projetados para poder ser configurados para as necessidades dos clientes de sistemas específicos |
| Bibliotecas de programas | Bibliotecas de classe e funções que implementam abstrações comumente usadas são disponibilizadas para reúso. |
| Engenharia dirigida a modelos | O software é representado como modelos de domínio e modelos de implementação independentes. O código é gerado a partir desses modelos. |
| Geradores de programas | Um sistema gerador incorpora o conhecimento de um tipo de aplicação, e é usado para gerar sistemas nesse domínio a partir de um modelo de sistema fornecido pelo usuário. |
| Desenvolvimento de software orientado a aspectos | Quando um programa é compilado, os componentes compartilhados são integrados em uma aplicação em diferentes locais. |

Fonte: Sommerville (2013).

Não faz parte do escopo deste trabalho esgotar o assunto de engenharia de software baseada em reuso, mas será enfatizada a abordagem de geradores de programas, aplicada no desenvolvimento do gerador de dashboards.

## 2.1.3 Geradores de programas

Uma definição consoante na literatura pesquisada é que geradores de programas são softwares que geram outros softwares a partir de especificações de alto nível (SOMMERVILLE, 2013). A programação automática ou generativa automatiza a produção de produtos intermediários ou finais, facilitando a implementação de processos trabalhosos e repetitivos, trazendo ganho de tempo e reduzindo a possibilidade de erro humano nessas implementações (LUCRÉDIO, 2009). Compiladores e geradores de interfaces gráficas embarcados em IDEs de mercado são exemplos de geradores de programas, que recebem especificações em um nível mais alto de abstração e como saída geram código fonte ou aplicações de mais baixo nível.

O princípio básico da programação generativa, em resumo, é garantir que o usuário especifique o que espera de um programa e que um software gere automaticamente o programa sem nenhuma assistência do usuário (SYRIANI; LUHUNU; SAHRAOUI, 2018).

A programação generativa pode ser utilizada em diversas fases do ciclo de vida do software. Gerar casos de testes, telas, relatórios ou até aplicações completas estão entre as possibilidades (LUCRÉDIO, 2009).

Um elemento chave na abordagem generativa é a forma de entrada que será fornecida ao gerador. Geralmente se utiliza uma linguagem específica de domínio, ou DSL. No caso de compiladores, a entrada passa a ser código-fonte em uma linguagem de alto nível, como Java. Algumas ferramentas CASE recebem especificações ou diagramas como entrada (SHIMABUKURO JUNIOR, 2006). Outra possibilidade é o uso de gabaritos ou *templates*. Gabaritos consistem em partes parcialmente prontas do produto, com marcações que são substituídas por parâmetros fornecidos ao gerador, que faz a composição, gerando o produto final concluído (LUCRÉDIO, 2009).

Um problema a ser considerado na geração de programas é quando se faz necessário alterar o produto gerado. Por mais direcionado ao domínio de aplicação e abstrato que o gerador possa ser, é bem possível que sejam necessárias algumas personalizações nos produtos gerados. Se a saída do gerador for código fonte aberto, é muito menos trabalhoso fazer as alterações diretas no código gerado. Alterações no gerador, por sua tendência a ser dedicado a um domínio de aplicação específico, podem requerer uma análise cuidadosa, pois há risco de que as novas versões de produtos gerados sejam incompatíveis com esse domínio (LUCRÉDIO, 2009). Já modificações feitas diretamente no código gerado podem ser perdidas se executarmos novamente o gerador, mesmo que usemos as mesmas especificações de entrada. Para evitar a perda de informações modificadas manualmente nos produtos de saída, o desenvolvedor do software deve fazer um controle adequado de versões do código gerado (SHIMABUKURO JUNIOR, 2006).

## 2.2 Motores de gabarito (*Template Engines*)

Motores de gabarito, também conhecidos como processadores de gabaritos ou analisadores de gabaritos, consistem em partes ou componentes de software que têm a função de combinar um ou mais gabaritos com um dado modelo de dados, gerando um ou mais artefatos de saída como resultado de seu processamento(WIKIPEDIA.ORG, 2020). Estes artefatos de saída podem ser desde um texto formatado simples, até um código fonte complexo.

Os gabaritos desenvolvidos para processamento são arquivos com conteúdo estático entremeado com marcações da linguagem específica de gabarito do processador em uso. São representações abstratas e generalizadas da saída textual que representam (SYRIANI; LUHUNU; SAHRAOUI, 2018). É comum os processadores de gabarito atuais contarem com recursos parecidos com os constantes em linguagens de alto nível, além de marcações de formatação. Variáveis, funções, inclusões de arquivos, Estruturas de decisão e laços fazem parte das ferramentas disponíveis (WIKIPEDIA.ORG, 2020).

## 2.2.1 Elementos básicos de motores de gabarito

Todos os processadores de gabarito são compostos ao menos de um modelo de dados associado, um ou vários gabaritos fonte, o motor de composição e, por fim, documentos de saída processados (WIKIPEDIA.ORG, 2020).

O modelo de dados pode ser um banco de dados relacional, um arquivo XML, uma planilha ou qualquer fonte de dados pré-formatados. Alguns processadores têm restrições quanto aos tipos de dados que podem ser utilizados, apesar de os mais utilizados serem desenvolvidos para permitir uma maior flexibilidade. É possível que o processador requeira do usuário algumas especificações além do modelo de dados, que definem alguma diferenciação na saída processada. Ambos podem ser chamados de entradas em tempo-de-execução (Runtime inputs) (SYRIANI; LUHUNU; SAHRAOUI, 2018).

Gabaritos fonte ou de origem consistem normalmente em texto com marcações em uma linguagem de gabaritos especialmente definida, também chamadas de entradas em tempo-de-desenvolvimento (Design-time inputs), que podem assemelhar-se a funções ou métodos de linguagens de programação (WIKIPEDIA.ORG, 2020).

O motor de composição é responsável por conectar o modelo de dados, receber especificações que podem ser passadas pelo usuário em tempo de execução, processar o gabarito, substituindo as marcações pelos dados ou especificações alimentadas e direcionar a saída para um documento de saída em disco, um vetor ou coleção na memória ou um fluxo de dados que gere a exibição instantânea numa tela, impressão ou, por exemplo, um navegador web (WIKIPEDIA.ORG, 2020). A figura 2 mostra uma representação pictórica de um processador de gabaritos.

**Figura 2 – Representação de um motor ou processador de gabaritos**



**Fonte: Elaborada pelo autor.**

Dada a flexibilidade dos processadores de gabaritos, há o emprego deste tipo de ferramenta de software em diferentes aplicações. A revisão da literatura indica um uso maior no campo de desenvolvimento web, facilitando o emprego do padrão arquitetural Modelo-Visão-Controle, separando o código fonte em camadas bem distintas. Frameworks de desenvolvimento web disponíveis no mercado atual usam sua própria abordagem para processar as saídas para o usuário. Desenvolvedores em linguagem PHP têm disponível, entre outras ferramentas o Twig. Desenvolvedores Java têm a possibilidade de optar pelo uso de ferramentas, como o Apache Velocity, Apache Freemarker. No presente trabalho vamos nos ater a uma descrição breve dos três motores de gabaritos aqui descritos, tendo sido definido para uso no protótipo o motor Apache Freemarker, merecendo este, portanto, uma descrição mais ampla.

## 2.3 Motores de gabaritos disponíveis para utilização no mercado atual

Para melhor fundamentar o desenvolvimento do protótipo e a decisão de qual a melhor abordagem para o projeto do gerador de Dashboards, foi necessária uma pesquisa das ferramentas disponíveis no mercado, usando como parâmetro de seleção suas vantagens e desvantagens em relação ao desempenho, facilidade de uso e compatibilidade, data de lançamento da última versão estável à data do trabalho e se o projeto fornecia licenciamento gratuito e código-fonte aberto. Nas subseções seguintes descrevo os componentes estudados.

### 2.3.1 Apache Velocity

Velocity é um motor de gabaritos baseado em Java. Permite a qualquer um usar uma linguagem de gabaritos simples, porém poderosa, para referenciar objetos definidos em código Java (APACHE.ORG, 2019). A ferramenta visa garantir uma clara separação entre as camadas de apresentação e de negócios em aplicações web (WIKIPEDIA.ORG, 2019). É um projeto de código fonte aberto hospedado pela Apache Software Foundation e liberada sob a Licença Apache (WIKIPEDIA.ORG, 2019).

Usando Velocity, é possível renderizar dados de Objetos concretos Java para texto, XML, HTML ou qualquer tipo de saída pré-formatada. O motor Velocity recebe como entrada o gabarito com as marcações e um objeto Java com dados brutos acessíveis por meio de uma interface pública (CRUZ; MOURA, 2002), que define o contexto de utilização dos dados. Essa definição de contexto faz o mapeamento entre as marcações no gabarito e a saída composta e devidamente formatada (MOURA *et al.*, 2004).

A linguagem específica do Velocity é a VTL (Velocity Template Language). Como possui especificações muito simples, as marcações são de fácil assimilação sendo agrupadas em referências, sempre iniciadas com caractere $ e diretivas, sempre precedidas do caractere #.

As referências são usadas para obter valores associados a elas, enquanto as diretivas são utilizadas para manipular a saída baseada nas informações obtidas do objeto Java (BAELDUNG.COM, 2017).

Os 3 tipos de referências do Velocity são:

* Variáveis: Podem ser definidas diretamente no gabarito utilizando a diretiva #set ($variável=valor). Podem receber valores diretamente do objeto Java;
* Propriedades: São referências a atributos do objeto Java. Podem também fazer referência a um método *getter* do atributo. A definição no gabarito deve ser feita da forma $objeto.atributo;
* Métodos: Devem ser definidos de forma semelhante às propriedades, trazendo o valor retornado pelo método acessado, na forma $objeto.metodo();

O resultado no documento final de saída de uma referência será sempre o valor obtido convertido em uma cadeia de caracteres ou uma string (BAELDUNG.COM, 2017).

A VTL possui um conjunto bastante completo de diretivas, garantindo a máxima flexibilidade na aplicação dos gabaritos em diversos tipos de domínios. A Tabela 2 traz uma listagem resumida contendo as diretivas e suas funcionalidades, bem como um exemplo simples de como utilizar a sintaxe da linguagem.

**Tabela 2 - Resumo de diretivas Apache Velocity VTL**

****

**Fonte: (XNAT.ORG, 2019)**

O Apache Velocity é licenciado sob a Apache License 2.0, que é totalmente compatível com a Gnu Public License 3.0. Isto garante a liberdade de uso e modificação do código fonte, bem como sua distribuição, desde que citada a origem na documentação do software final.

### 2.3.2 Apache Freemarker

Apache Freemarker é um motor de gabaritos para Java, tendo sua versão inicial disponibilizada em 2000, desenvolvida por Benjamin Geer e Mike Bayer (WIKIPEDIA.ORG, 2018). Consiste em uma biblioteca Java capaz de gerar saídas de texto baseadas em um ou mais gabaritos e um conjunto de dados variáveis em tempo de execução. Esta saída pode ser página HTML, texto formatado, código fonte, arquivos de configuração ou scripts em geral (APACHE.ORG, 2020). Os gabaritos para uso no motor de composição do Freemarker são escritos usando uma linguagem especializada chamada FTL (Freemarker Template Language), que possui um dicionário de marcações a serem entremeadas ao conteúdo estático.

Apesar de ter sido desenvolvido com a finalidade de desenvolvimento de páginas HTML em frameworks que adotassem o padrão arquitetural MVC (Modelo-Visão-Controle), a versatilidade de gerar saídas em texto plano garante a aplicabilidade e múltiplos domínios para o motor Apache Freemarker (APACHE.ORG, 2020).

A FTL, bem como a VTL possui funcionalidades como blocos condicionais, laços, iteradores para listas ou coleções, operações aritméticas, operações e formatação para cadeias de caracteres além de macros. A sintaxe básica das marcações é ${atributo}. De forma resumida, um gabarito FTL é composto de:

* Texto estático;
* Interpolações: Seções que serão substituídas em tempo de execução com atributos e valores do modelo de dados. São delimitadas por ${ e };
* Tags FTL: Descritas no gabarito como tags HTML, delimitadas por <# e >, porém são instruções ao motor Freemarker, não sendo exibidas no documento de saída;
* Comentários: Descrições com finalidade de documentação, delimitadas por <#-- e -->. Assim como as Tags FTL, serão ignorados pelo Motor Freemarker e não serão exibidos na composição do documento de saída.

De forma resumida, seguem abaixo descritas as regras básicas de expressões em gabaritos Freemarker (APACHE.ORG, 2020).

Valores especificados diretamente

* Strings: *"Valor"* ou *'Valor'* ou *"Entre \"Aspas\""* ou *r"C:\raw\string"*
* Números: 123.45
* Booleanos: true, false
* Sequencias: *["foo", "bar", 123.45];* Intervalos: *0..9, 0..<10 (*ou *0..!10), 0..*
* Hashes: *{"name":"green mouse", "price":150}*

Acessando Variáveis

* Primeiro Nível: *user*
* Hash: *user.name* ou *user["name"]*
* Sequencias: *products[5]*
* Variável Especial: *.main*

Operações com Strings

* Interpolação e Concatenação : *"Hello ${user}!"* (or  *"Hello " + user + "!"*)
* Obtendo um caractere em uma String: *name[0]*
* Parte de uma String: Final inclusivo: *name[0..4]*, Final exclusivo: *name[0..<5]*, Baseado no comprimento: *name[0..\*5]*, Removendo inicio: *name[5..]*

Operações com sequências

* Concatenação: *users + ["guest"]*
* Parte de uma sequencia: Final inclusivo: *products[20..29],* Final Exclusivo: *products[20..<30]*, Baseado no comprimento: *products[20..\*10],* Remove inicio: *products[20..]*

Operações com hashes

* Concatenação: *passwords + { "joe": "secret42" }*

Calculo Aritmético: (x \* 1.5 + 10) / 2 - y % 100

Comparações: *x == y, x != y, x < y, x > y, x >= y, x <= y, x lt y, x lte y, x gt y, x gte y, ...etc.*

Operações Lógicas : *!registered && (firstVisit || fromEurope)*

Funções Embutidas: *name?upper\_case, path?ensure\_starts\_with(‘/’)*

Chamadas de método: *repeat("What", 3)*

Operador de valores perdidos (*null* ou *void*):

* Valor Padrão: *name!"unknown"* or *(user.name)!"unknown"* or *name!* or *(user.name)!*
* Teste de valor desconhecido: name?? or (user.name)??

Operadores de atribuição: *=, +=, -=, \*=, /=, %=, ++, --*

Lambdas locais: *x -> x + 1, (x, y) -> x + y*

# 3 - Desenvolvimento do Protótipo

Neste capítulo é apresentado o desenvolvimento da ferramenta de geração de dashboards, utilizando práticas de análise orientada a objetos, além de demonstrar exemplos de aplicação e os resultados obtidos.

## 3.1 - Descrição Geral

Partindo de uma especificação de um arquivo de intercambio de dados CSV, a aplicação IFPA DashGen permite gerar um Dashboard contendo uma tabela e 3 tipos de gráficos, sendo 1 gráfico de área, 1 gráfico de barras e 1 gráfico pizza, com filtros dinâmicos que permitem seleção das dimensões pelo usuário, assim como redefinição para a visão inicial.

O funcionamento da aplicação se dá como segue descrito abaixo:

1. O usuário indica ao DashGen qual o arquivo fonte de dados;
2. O DashGen captura a primeira linha do arquivo fonte de dados, considerando que esta traga os nomes dos atributos contidos;
3. O usuário especifica o domínio (eixo X) e a imagem(Eixo Y) de cada tipo de gráfico, bem como o título e as etiquetas dos atributos de domínio e imagem;
4. O usuário especifica o caminho onde o arquivo contendo a estrutura do Dashboard deve ser armazenado;
5. O DashGen instancia os Templates incluindo os dados especificados, gerando um Dashboard completo.
6. O DashGen salva um arquivo compactado em formato ZIP na pasta especificada pelo usuário, que pode ser descompactado no caminho da aplicação que este desejar.

(*fazer figura ilustrando os passos descritos)*

## 3.2 Descrição das Etapas de Desenvolvimento

O trabalho de desenvolvimento da aplicação IFPA DashGen foi dividido nos seguintes passos:

1. Levantamento dos requisitos;
2. Especificação do uso do motor de Templates, baseado nos diagramas UML de caso de uso, de classes e de sequencia;
3. Implementação do motor de Templates;
4. Implementação do gerador de Dashboard.

## 3.3 – Levantamento dos Requisitos

A aplicação produzida neste trabalho, a partir das especificações de entrada, será capaz de gerar um Dashboard completo. As entradas são: o arquivo de dados, os arquivos de template, os atributos de dimensão, o título do arquivo e o caminho para armazenamento do Dashboard gerado.

Baseado nisso, foi feito o levantamento de requisitos a serem atendidos, fechando o escopo da aplicação. No quadro 1, podem ser observados os requisitos funcionais e no quadro 2, podem ser observados os requisitos não-funcionais.

|  |  |  |
| --- | --- | --- |
| **Requisitos Funcionais** | | |
| **Requisito** | **Descrição** | **Caso de Uso** |
| RF01 | O sistema deve acessar o arquivo de dados CSV | UC01 |
| RF02 | O sistema deve permitir que o usuário especifique o arquivo de dados | UC01 |
| RF03 | O sistema deve identificar os metadados do arquivo de dados | UC01 |
| RF04 | O sistema deve ser capaz de identificar atributos numéricos dentro dos conjuntos de dados | UC01 |
| RF05 | O sistema deve permitir que o usuário especifique o caminho onde o Dashboard deve ser armazenado | UC01 |
| RF06 | O sistema deve gerar o Dashboard, compactando toda a estrutura em um arquivo ZIP, no caminho especificado pelo usuário. | UC01 |

Quadro 1 – Requisitos Funcionais

|  |  |
| --- | --- |
| **Requisitos não funcionais** | |
| **Requisito** | **Descrição** |
| RNF01 | Sistema deve ser desenvolvido em Java 8 SE. |
| RNF02 | Sistema deve ter interface gráfica simples e intuitiva, com uso de mouse, tendo todas as informações exibidas com clareza e ter funcionalidades acessíveis por botões e menus. |
| RNF03 | O desempenho da geração do Dashboard deve ser alto. O tempo de geração do arquivo destino não deve exceder 30 segundos. |
| RNF04 | O padrão do arquivo de dados deve ser CSV, com a primeira linha contendo a descrição dos atributos. |
| RNF04 | A linguagem de templates a ser aplicada será a Apache FreeMarker. |
| RNF05 | O formato de compactação do arquivo de saída será ZIP. |

Quadro 2 – Requisitos Não Funcionais

## 3.4 – Casos de Uso

Na interação com o usuário, a aplicação terá um único caso de uso, como pode ser verificado no diagrama de casos de uso.



Figura 1 – Caso de uso UC01

A descrição do caso de uso UC01 – Gerar\_Dashboard foi detalhada considerando o processo completo de geração do Dashboard do ponto de vista do usuário da aplicação. Este detalhamento pode ser observado no Quadro 3.

|  |  |
| --- | --- |
| **UC01 – Caso de uso Gerar\_Dashboard:** Permite ao usuário especificar o arquivo de dados, os atributos a serem usados como dimensões nos gráficos e o caminho para a geração do arquivo compactado com o Dashboard completo. | |
| Pré-condições: | Usuário executa a aplicação Dashgen. |
| Cenário Principal: | 1. Ao acessar o sistema, o usuário informa o caminho para o arquivo CSV; 2. O sistema apresenta o primeiro gráfico e solicita que sejam indicadas as dimensões, apresentando os atributos obtidos da primeira linha do CSV; 3. O sistema segue para a apresentação do segundo gráfico e solicita que sejam indicadas as dimensões, apresentando atributos obtidos do CSV; 4. O sistema então apresenta o terceiro gráfico, solicitando novamente as dimensões a partir dos atributos obtidos do CSV; 5. O caso de uso termina no momento em que o usuário faz o download do arquivo ZIP gerado pelo sistema. |
|  |  |
|  |  |

# 5 REFERÊNCIAS

APACHE.ORG. **The Apache Velocity Project**. *[s. l.]*, 2019. Disponível em: https://velocity.apache.org/. Acesso em: 7 jan. 2019.

APACHE.ORG. **FreeMarker Java Template Engine**. *[s. l.]*, 2020. Disponível em: https://freemarker.apache.org/index.html. Acesso em: 1 jun. 2019.

BAELDUNG.COM. **Introduction to Apache Velocity**. *[s. l.]*, 2017. Disponível em: https://www.baeldung.com/apache-velocity. Acesso em: 7 jan. 2019.

CRUZ, Sérgio Aparecido Braga; MOURA, Maria Fernanda. **Formatação de Dados Usando a Ferramenta Velocity**. 2002. *[s. l.]*, 2002.

KRUEGER, Charles W. Software reuse. **ACM Computing Surveys**, *[S. l.]*, v. 24, n. 2, p. 131–183, 1992. Disponível em: https://www.cs.ait.ac.th/~on/O/oreilly/perl/advprog/ch17\_01.htm

LUCRÉDIO, Daniel. Uma Abordagem Orientada a Modelos para Reutilização de Software. *[S. l.]*, p. 277, 2009.

MOURA, Maria Fernanda *et al.* **Comunicado Técnico Uma Análise Comparativa das Soluções Tecnológicas Utilizadas nas Apresentações de Dados da Agência de Informação Embrapa**. 2004. - Embrapa, *[s. l.]*, 2004.

SHIMABUKURO JUNIOR, Edison Kicho. Um Gerador de aplicações configurável. *[S. l.]*, 2006.

SOMMERVILLE, Ian. **Engenharia de Software**. 3. ed. Sao Paulo: Pearson, 2013.

SYRIANI, Eugene; LUHUNU, Lechanceux; SAHRAOUI, Houari. Systematic mapping study of template-based code generation. **Computer Languages, Systems and Structures**, *[S. l.]*, v. 52, p. 43–62, 2018.

WIKIPEDIA.ORG. **Apache FreeMarker**. *[s. l.]*, 2018. Disponível em: https://en.wikipedia.org/wiki/Apache\_FreeMarker. Acesso em: 25 nov. 2018.

WIKIPEDIA.ORG. **Apache Velocity**. *[s. l.]*, 2019. Disponível em: https://en.wikipedia.org/wiki/Apache\_Velocity.

WIKIPEDIA.ORG. **Template Processor**. *[s. l.]*, 2020. Disponível em: https://en.wikipedia.org/wiki/Template\_processor. Acesso em: 6 fev. 2020.

XNAT.ORG. **Apache Velocity Cheatsheet**. *[s. l.]*, 2019. Disponível em: https://wiki.xnat.org/docs16/4-developer-documentation/xnat-codex/velocity-cheat-sheet. Acesso em: 7 ago. 2019.