INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PARÁ
CURSO DE GRADUAÇÃO TECNOLÓGICA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GLAUBER MATTEIS GADELHA

DashGen: Gerador de quadros de apresentação de dados em formato de *dashboards*

BELÉM
2020

GLAUBER MATTEIS GADELHA

DashGen: 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 Grau em Tecnologia em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Claudio Roberto de Lima Martins

BELÉM
2020

(RESERVADA PARA FICHA CATALOGRÁFICA)

GLAUBER MATTEIS GADELHA

DashGen: 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 Grau em Tecnologia em Análise e Desenvolvimento de Sistemas.

Data da Defesa: 12/03/2020
Conceito: ___

_____
Orientador: Prof. Me. Claudio Roberto de Lima Martins
Instituto Federal do Pará – Campus Belém

_____
Prof. Me. Délcio Nonato Araújo da Silva
Instituto Federal do Pará – Campus Belém

_____
Prof. Me. Andracir Alves Oliveira
Instituto Federal do Pará – Campus Belém

À Simone, minha esposa e companheira de todas as horas.

AGRADECIMENTOS

A Deus, pois é digno de toda Honra, toda Glória e todo Louvor, hoje e sempre.

A minha esposa e filhos pelo amor e pela compreensão em relação ao tempo em que precisei estar ausente para me dedicar ao meu emprego, à graduação e, por fim, a este trabalho de conclusão de curso.

Ao Prof. Me. Claudio Roberto de Lima Martins pela orientação, suporte e sugestões valiosas, não só para este trabalho, mas pelo decorrer do curso de Tecnologia em Análise e Desenvolvimento de Sistemas.

Aos mestres que passaram pela minha vida desde 2015 até 2020, no bloco C do Instituto Federal do Pará. Seu apoio e ensinamentos me serão úteis por toda minha vida.

A todos os amigos e colegas de trabalho da Fundação Cultural do Estado do Pará pelo apoio incondicional nos momentos em que precisei de ajuda em função do curso superior ou de problemas pessoais, em especial a Cezar Martins, Humberto Spindola, Nonato Ramos, Tatiane Launé, Rosa Oliveira e Dina Oliveira.

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 apresenta um estudo de reuso de software e geração de código fonte baseada em gabaritos, com desenvolvimento de uma aplicação (denominada DashGen) 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 code blocks, with the modifications necessary inside the domain of a specific problem, can avoid 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 (named DashGen) that generates a data presentation chart in format that we commonly call *Dashboards*.

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

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API - Application Programming Interface
ASP - Active Server Pages
CASE - Computer-Aided Software Engineering
COTS - Commercial off-the-shelf
CSS – Cascade Style Sheets
CSV – Comma Separated Values
DOM – Document Object Model
DSL – Domain Specific Language
ERP – Enterprise Resource Planning
FTL – FreeMarker Template Language
FXML – JavaFX Markup Language
GUI – Graphical User Interface
HTML – Hypertext Markup Language
IDE – Integrated Development Environment
JDK – Java Development Kit
JSON – Javascript Object Notation
JSP - Java Server Pages
MVC – Model-View-Controller
PHP – PHP Hypertext Processor
POM – Project Object Model
SVG – Scalable Vector Graphics
UML – Unified Markup Language
VTL – Velocity Template Language
WYSIWYG – What you see is what you get
XML – Extensible Markup Language
ZIP – formato de compactação de arquivos

SUMÁRIO

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 (COSTA et al., 2012). 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 painéis ou quadros gráficos, chamados *dashboards*, que entregam um resumo visual de centenas, muitas vezes milhares de linhas de um conjunto de dados. Tal recurso facilita o entendimento e suporta de forma mais simples e direta a tomada de decisão (LUIZ, 2013).

Mesmo com a evolução das linguagens de programação além da expansão de suas bibliotecas de Interfaces de Aplicações para Programação, as chamadas APIs, ainda é trabalhoso desenvolver a camada de apresentação de dados de forma gráfica e dinâmica. Uma página web em tecnologias como JSP[1], ASP.NET[2] ou até mesmo HTML com JavaScript, contendo um *dashboard* que sintetiza 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 para cada domínio de problema. (LUCRÉDIO, 2009).

[1] *JavaServer Pages* (JSP) é uma tecnologia que suporta a criação de páginas web geradas dinamicamente em HTML, mas usando a linguagem de programação Java.
[2] ASP.NET é a plataforma da Microsoft para o desenvolvimento de aplicações Web. È o sucessor da tecnologia ASP. Permite, através de uma linguagem de programação integrada na arquitetura .NET Framework, criar páginas dinâmicas.

Estratégias como utilizar-se de bibliotecas, *frameworks* e algoritmos que gerem código a partir de um conjunto de definições feitas previamente pelo usuário, 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, aqui denominada **DashGen**, cujo objetivo é receber 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.

1.1 Objetivo e delimitação de escopo

O objetivo deste trabalho é desenvolver uma aplicação em linguagem Java 8 SE para *desktops*, usando técnicas de programação generativa orientada a gabaritos (no inglês, *templates*) que facilite a criação de painéis de informações em formato de gráficos e tabelas (*dashboards*), gerados a partir de dados obtidos de um arquivo CSV (Comma Separated Values, ou Valores separados por vírgula).

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 (b) redução por somatória, usando um atributo especificado como chave e a totalização de um segundo atributo especificado como um valor numérico.

Neste trabalho, define-se **gabarito** (ou *template*) como um arquivo contendo marcações HTML e *scripts* em código Javascript, 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, que se

encarregam de aplicar os filtros dinâmicos e gerar os painéis com gráficos em formato SVG do HTML5.

A interface gráfica de interação entre o usuário e a aplicação foi desenvolvida usando JavaFX 8.

1.2 Procedimentos metodológicos

O trabalho foi conduzido em três etapas. No primeiro momento foi feita a investigação da literatura relacionada a fim fundamentar teoricamente a solução proposta, abordando temas como 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 do projeto da aplicação, utilizando-se de algumas técnicas já consolidadas da engenharia de software.

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

1.3 Organização do trabalho

Este trabalho de conclusão de curso foi organizado da seguinte maneira. 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, são apresentadas soluções de software correlatas à aplicação proposta neste trabalho.

No capítulo 4, apresenta-se o projeto da aplicação geradora de *dashboards* e seu desenvolvimento. Por último, no capítulo 5 são feitas as considerações finais e propostas para trabalhos futuros.

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 de painéis (*dashboard*). Além disso, fundamenta-se as bibliotecas e tecnologias utilizadas no desenvolvimento da aplicação final.

2.1 Reuso de Software

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 ao ano de 1947, no início da programação dos computadores, quando Wheeler e Wilkes desenvolveram o conceito de *jump*, um precursor do comando *goto*, que possibilitava reaproveitar blocos de código dentro do mesmo programa. A partir deste momento, programadores passam a reaproveitar 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, que viabilizados pela 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 e contexto. Reaproveitar artefatos devidamente testados e validados certamente pode trazer benefícios na redução do custo total do desenvolvimento de

um sistema, entre outras vantagens, porém 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):

a) 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.
b) Aumento da confiança: reutilizar soluções previamente desenvolvidas, testadas e validadas geralmente reflete uma confiabilidade maior do que uma nova solução desenvolvida.
c) 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):

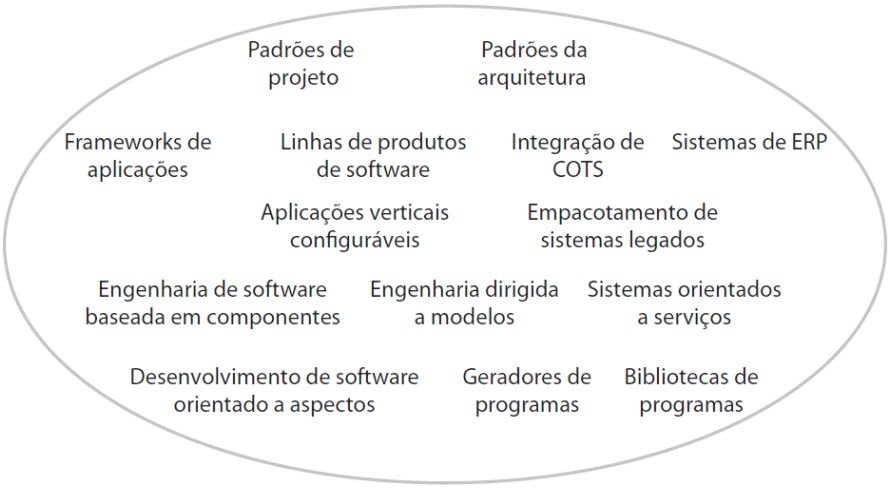
a) Ausência de ferramentas de suporte: certas ferramentas não dão suporte adequado ao reuso. Isso pode dificultar ou até mesmo impedir o emprego destas ferramentas em um novo sistema.
b) Síndrome do “não inventado aqui”: o desenvolvedor pode decidir não usar uma ferramenta pronta pela dificuldade de entendê-la ou por se achar capaz de fazer melhor.
c) 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.

2.1.1 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).

Na Figura 1 apresentam-se 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*.

2.1.2 Geradores de programas

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 geração 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). Em ambientes integrados de desenvolvimento (em inglês, IDEs) é comum encontrar entre os seus recursos 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 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 (do inglês *Computer-Aided Software Engineering*) recebem especificações ou diagramas como entrada (SHIMABUKURO JUNIOR, 2006). Outra possibilidade é o uso de gabaritos ou *templates*. Gabaritos consistem em conteúdos parcialmente prontos 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

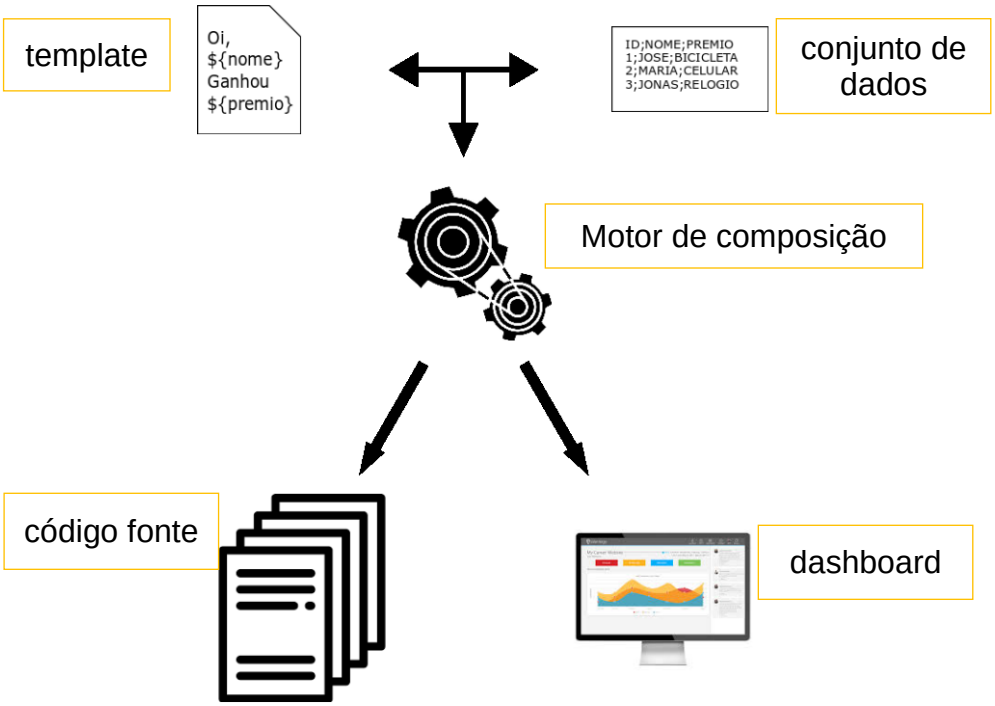
Todos os processadores de gabarito são compostos ao menos de um modelo de dados associado, um ou vários gabaritos fonte, o motor de composição e, por fim, documentos de saída processados (WIKIPEDIA.ORG, 2020).

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, embora os mais utilizados sejam implementados pelas ferramentas 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 para esse propósito, são 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 processar o gabarito substituindo as marcações pelos dados ou especificações alimentadas e direcionar a saída para um documento final em disco, um vetor na memória ou um fluxo de dados que gere a exibição instantânea numa tela (WIKIPEDIA.ORG, 2020). Na Figura 2 pode-se observar 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. Na literatura, observa-se um uso maior no campo de desenvolvimento web, facilitando o emprego do padrão arquitetural Modelo-Visão-Controle (MVC), 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 (POTENCIER, 2019). Desenvolvedores Java têm a possibilidade de optar pelo uso de ferramentas, como o Apache Velocity (APACHE.ORG, 2019a) e Apache Freemarker (APACHE.ORG, 2020). No presente trabalho vamos nos ater a uma descrição breve do dois motores de gabaritos para aplicações Java aqui descritos, tendo sido definido para uso no protótipo o motor Apache Freemarker, merecendo este, portanto, uma descrição mais ampla.

2.3 Estado da arte em motores de gabaritos

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 descreve-se os componentes estudados.

2.3.1 Apache Velocity

Velocity é um motor de gabaritos baseado em Java. Permite usar uma linguagem de gabaritos simples, porém poderosa, para referenciar objetos definidos em código Java (APACHE.ORG, 2019a). A ferramenta visa garantir uma clara separação entre as camadas de apresentação e de negócios em aplicações web (WIKIPEDIA.ORG, 2019a). É um projeto de código fonte aberto iniciado em 2000 como parte do projeto Apache Jakarta, hospedado pela Apache Software Foundation e liberada sob a Licença Apache (WIKIPEDIA.ORG, 2019a).

Com Velocity é possível formatar 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, que define o contexto de utilização dos dados (CRUZ; MOURA, 2002). 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:

a) **Variáveis**: Podem ser definidas diretamente no gabarito utilizando a diretiva #set ($variável=valor). Podem receber valores diretamente do objeto Java.

b) **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.
c) **Métodos**: Devem ser definidos de forma semelhante às propriedades, trazendo o valor retornado pelo método acessado, na forma $objeto.método().

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 Figura 3 representa uma listagem resumida contendo as diretivas e suas funcionalidades, bem como um exemplo simples de como utilizar a sintaxe da linguagem.

Figura 3 - Resumo de diretivas Apache Velocity VTL

Notação variável	$
Comentários (uma linha)	##
Comentários (várias linhas)	#* *#
Operadores aritméticos	+, -, *, /, % (módulo)
Relacional (não apenas equivalência, pode ser usado para comparar objetos)	==
Operadores lógicos	&&, |, !
Operador de faixa (usado em loops)	[n..m]
Caractere de escape	\
Referenciar uma variável	$foo
Faça referência a uma variável (se o valor for nulo, imprima nada)	$!foo
Atribua um valor a uma variável (string literal)	#set ($!foo = "Velocity")
Consulte uma chave de hashtable ou um método get (Address)	$customer.address
Afirmação condicional	#if ($foo) Velocity! #elseif ($foo2 == "cool") XNAT! #end
Ciclo	#foreach ($criterion in $criteria) ## loop ArrayList Current Value: $!criterion #end
Incorporar texto de outro modelo	#parse ("parsefoo.vm")

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 (APACHE.ORG, 2019a).

2.3.2 Apache Freemarker

Apache Freemarker é um motor de gabaritos para aplicações Java, tendo sua versão inicial disponibilizada em 2000 pelos desenvolvedores 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 definido em tempo de execução. Esta saída pode ser uma 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:

a) **Texto estático:** qualquer conteúdo que se deseje que se mantenha fixo no documento final após a composição.
b) Interpolações: Seções que serão substituídas em tempo de execução com atributos e valores do modelo de dados. São delimitadas por `${` e `}`.
c) **Tags FTL**: Descritas no gabarito da mesma forma que tags HTML, delimitadas por <# e >, porém são instruções ao motor Freemarker, não sendo exibidas no documento de saída.
d) **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, as regras básicas de expressões em gabaritos Freemarker (APACHE.ORG, 2020) possuem as seguintes regras.

a) 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}*

b) Acesso às variáveis

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

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

d) Operações com hashes

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

e) **Cálculo, lógica e outras operações**

- 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*

Quanto às características técnicas, o Apache Freemarker é mais rígido do que o Apache Velocity e outros motores de gabaritos. Retorna exceções de ponteiro nulo (*null pointer exceptions*) se encontra uma propriedade inexistente ao processar o gabarito. Tem funcionalidades de macro mais avançadas e fornece mais recurso aos desenvolvedores da camada de apresentação do que o Apache Velocity, como operações avançadas com datas e horas. (BERGEN, 2007).

O Apache Freemarker é também 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 (APACHE.ORG, 2020).

2.3.3 Critérios usados na avaliação dos motores de gabaritos pesquisados

Para que fosse possível decidir tecnicamente qual o motor de gabaritos seria utilizado no projeto objeto deste trabalho de conclusão de curso, foram avaliadas as características descritas na Tabela 2. Os dados foram obtidos em uma pesquisa na ferramenta de busca *google.com* e em alguns artigos encontrados ao longo do período de levantamento bibliográfico.

Tabela 2 - Comparativo Apache Velocity X Apache Freemarker

ITEM	Velocity	Freemarker
Pesquisa no google.com (Número aproximado de resultados) em junho de 2019	**6.222.000**	647.000
Popularidade no mavenrepository.org (Número de requisições por artefato) em junho de 2019	1427	**1766**
Tempo de finalização de processamento de gabarito simples Olá Mundo!, medido por chamada *System.currentTimeMillis()* (BERGEN, 2007)	141	**110**
Tempo de finalização de processamento de gabarito complexo, medido por chamada *System.currentTimeMillis()* (BERGEN, 2007)	32317	**11647**
Data do lançamento da ultima versão estável em agosto de 2019. (APACHE.ORG, 2019b, 2019c)	31/03/2019 V2.1	**17/08/2019** **V2.3.29**
Data do lançamento da penúltima versão estável. (APACHE.ORG, 2019b, 2019c)	06/08/2017 V2.0	**04/04/2018** **V2.3.28**

Fonte: Elaborada pelo autor.

A escolha recaiu na ferramenta *Apache Freemarker*, entre outros motivos porque tal tecnologia encontrava-se mais ativa nos últimos dois anos à data da pesquisa. Atributos de desempenho e popularidade (medidas num repositório do gerenciador de dependências Apache Maven) (MAVENREPOSITORY, 2019a, 2019b) foram também cruciais para a decisão de empregar o Apache Freemarker no projeto objeto do presente trabalho.

2.2 Apache Maven

O Apache Maven é um automatizador de compilação e gerenciador de dependências utilizado com muita frequência em projetos em linguagem Java, mas não limitado somente a esta, podendo ser também utilizado em projetos C#, Ruby e SCALA (WIKIPEDIA.ORG, 2019b). É uma ferramenta de gerenciamento e compreensão de projeto. Baseado em um Modelo de Projeto de Objeto (Project Object Model – POM), o Maven pode gerenciar a documentação, relatórios e compilação a partir de uma peça central de informação (APACHE.ORG, 2019d).

O Maven tem como objetivos principais (APACHE.ORG, 2019d):

a) Facilitar o processo de construção e compilação.

b) Prover um sistema uniforme de construção.
c) Prover informação do projeto com qualidade.
d) Prover diretivas para melhores práticas de desenvolvimento.
e) Garantir a migração transparente de novas funcionalidades.

O Modelo de Projeto de Objeto é armazenado em um arquivo XML denominado **pom.xml**. Na Figura 4, visualiza-se um exemplo da estrutura mínima do POM.

Figura 4 – Exemplo de configuração do arquivo pom.xml

```
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>

  <!-- project coordinates, i.e. a group of values which
       uniquely identify this project -->

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- library dependencies -->

  <dependencies>
    <dependency>

      <!-- coordinates of the required library -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- this dependency is only used for running and compiling tests -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

Fonte: (WIKIPEDIA.ORG, 2019b)

Um projeto Maven tem uma estrutura padronizada de diretórios com as entradas como seguem demonstradas na Tabela 3.

Tabela 3 - Estrutura de diretórios padrão do Maven

NOME DO DIRETÓRIO	PROPÓSITO
/ (raiz do projeto)	Contém o pom.xml e todos os subdiretórios
src/main/java	Contém o código fonte e os pacotes do projeto Java
src/main/resources	Contém os recursos disponíveis para o projeto, como arquivos contendo propriedades
src/test/java	Contem os códigos fonte Java para testes
Src/test/resources	Contém os recursos necessários para os testes.

Fonte: (WIKIPEDIA.ORG, 2019b)

Uma das funcionalidades principais do Apache Maven é o gerenciamento de dependências. O mecanismo de gerenciamento é organizado em torno de um sistema de coordenadas que identifica artefatos individuais como bibliotecas de software ou módulos. As informações a respeito da dependência são inseridas no pom.xml na seção <dependencies>. Na Figura 5 observa-se uma declaração de dependência do Apache Freemarker.

Figura 5 – Exemplo de uma declaração de dependência no pom.xml

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.freemarker/freemarker -->
    <dependency>
        <groupId>org.freemarker</groupId>
        <artifactId>freemarker</artifactId>
        <version>2.3.28</version>
    </dependency>
</dependencies>
```

Fonte: Elaborada pelo autor

Com essa especificação feita no arquivo pom.xml, o Apache Maven consulta o repositório e faz o download do artefato identificado na versão especificada. Isso automatiza totalmente a busca pelas bibliotecas ou módulos necessários para o projeto, garantindo que todas as dependências estejam disponíveis para a próxima compilação.

2.4 DC.js

A DC.js é uma biblioteca feita em Javascript para desenho de quadros contendo gráficos para análise de dados, que se utiliza de outras duas bibliotecas abertas: a Crossfilter.js e a D3.js. A função da DC.js é facilitar a implementação de

gráficos com filtros dinâmicos no lado cliente de uma aplicação web, o que a torna extremamente veloz (TEAM DC.JS, 2018; TUTORIALSPOINT, 2018). Determinando os tipos de gráficos, as dimensões e as formas de filtro e redução, é possível gerar em uma página HTML5 gráficos dinâmicos contendo a síntese de milhares de linhas de uma fonte de dados, que pode ser um arquivo CSV, XML ou uma fonte de dados JSON. Nas subseções seguintes há uma descrição sucinta das dependências principais da DC.js.

2.4.1 Crossfilter.js

Crossfilter.js é uma biblioteca Javascript para exploração de grandes conjuntos de dados multivariados em um navegador web. Mesmo em conjuntos de dados com milhares de registros, suporta uma interação abaixo de 30 milissegundos com visualizações coordenadas (CROSSFILTER ORGANIZATION, 2018). Usa um conceito de filtragem e redução incrementais a partir do ajuste inicial de uma das dimensões determinadas, tornando ainda mais rápido o rearranjo das exibições do que se fossem iniciadas do zero (CROSSFILTER ORGANIZATION, 2018). A biblioteca inicial, mantida pela SQUARE possui uma bifurcação mantida pela comunidade, a qual será utilizada neste trabalho.

2.4.2 D3.js

D3.js é uma biblioteca Javascript desenvolvida com a finalidade de manipular documentos com base em dados. Ajuda a transformar dados em gráficos usando HTML, SVG e CSS. É totalmente baseada nos padrões da web para ser compatível com a maioria dos navegadores modernos, evitando a aderência a qualquer plataforma proprietária, combinando componentes poderosos para visualização e uma abordagem orientada a dados para a manipulação do DOM. D3.js é extremamente rápida e suporta grandes quantidades de dados e comportamentos dinâmicos que permitem interação e animação (BOSTOCK et al., 2018).

2.5 Outras bibliotecas utilizadas neste trabalho

Esta seção do trabalho descreve as demais bibliotecas utilizadas para a composição do projeto DashGen.

2.5.1 Apache Commons CSV

Parte do projeto Apache Commons, a biblioteca Commons CSV traz funcionalidades para leitura e escrita de arquivos com dados separados por vírgula. Traz suporte embutido para os mais comuns formatos de arquivos CSV: Microsoft Excel, Informix UNLOAD CSV, MySQL, Oracle, PostgreSQL CSV e Text, RFC 4180 e TDF (APACHE.ORG, 2019e).

A Commons CSV transforma os arquivos CSV em uma coleção iterável, permitindo o acesso e tratamento das informações, assim como a escrita de dados em arquivos formatados. Neste trabalho a biblioteca foi útil para fazer a coleta das etiquetas dos atributos contidas na primeira linha do arquivo, e também na iteração dos registros contidos para identificação de dados de tipo numérico.

2.5.2 Apache Commons IO

Apache Commons IO é uma biblioteca Java para facilitar a implementação de funcionalidades de entrada e saída de dados. É composta de classes utilitárias para tarefas comuns, classes de Entrada e Saída para implementação de fluxos de dados, Filtros para arquivos, comparadores e monitores de eventos (APACHE.ORG, 2019f)

No presente trabalho, foi utilizada para facilitar a implementação da rotina de cópia dos arquivos comuns para o diretório de saída do dashboard do usuário.

2.5.3 Zeroturnaround ZT-ZIP

A ZT-ZIP é uma biblioteca com funcionalidades de compactação de arquivos em formato ZIP. Foi criada por Rein Raudjärv para facilitar a implementação dos pacotes java.util.zip (RAUDJÄRV, 2019). Torna a compactação ou descompactação de diretórios recursivamente muito mais simples. Permite também incluir ou excluir entradas nos arquivos compactados, iterar entre as entradas do arquivo e comparar dois arquivos no formato ZIP. É compatível com o Java a partir da versão 1.5 (RAUDJÄRV, 2019).

2.6 JavaFX 8

O JavaFX foi criado como uma plataforma de software para criar e fornecer aplicativos para aplicações desktop e web, que podem ser executados em múltiplas plataformas e dispositivos (WIKIPEDIA.ORG, 2019c). É um conjunto de pacotes de gráficos e mídia que dão a possibilidade do desenvolvedor criar aplicações cliente com uma experiência rica de interface (ORACLE, 2019).

O primeiro lançamento de versão do JavaFX se deu em 4 de dezembro de 2008, com a liberação da versão 1.0.2, quando o Java ainda era de propriedade da Sun Microsystems. A intenção era de que o JavaFX substituísse completamente o Swing como o conjunto padrão de ferramentas para interface gráfica de usuário ou GUI. Na versão 8u241 do Java Development Kit (JDK), ambos os frameworks estão incorporados à API do Java 8 SE.

Desde a liberação do JDK 11 no ano de 2018, o JavaFX tornou-se parte do OpenJDK, sob o nome de projeto OpenJFX, porém, a Oracle, atual proprietária do Java SE, manterá o suporte *premium* ao JavaFX 8 até Março de 2022 (WIKIPEDIA.ORG, 2019c). Portanto, para evitar problemas de compatibilidade e de implementação do OpenJFX, foi escolhido para o projeto deste trabalho a versão da Oracle do JDK 8 (Versão 8u241) .

Aplicações em JavaFX seguem o mesmo conceito defendido na plataforma Java: *“Write Once, Run Everywhere”* (desenvolva uma vez, execute em qualquer lugar). Considerando o fato de ser parte da API do JavaSE desde a versão 7, qualquer plataforma que possua o Java Runtime Environment (JRE) pode executar a aplicação sem necessidade de qualquer software intermediário.

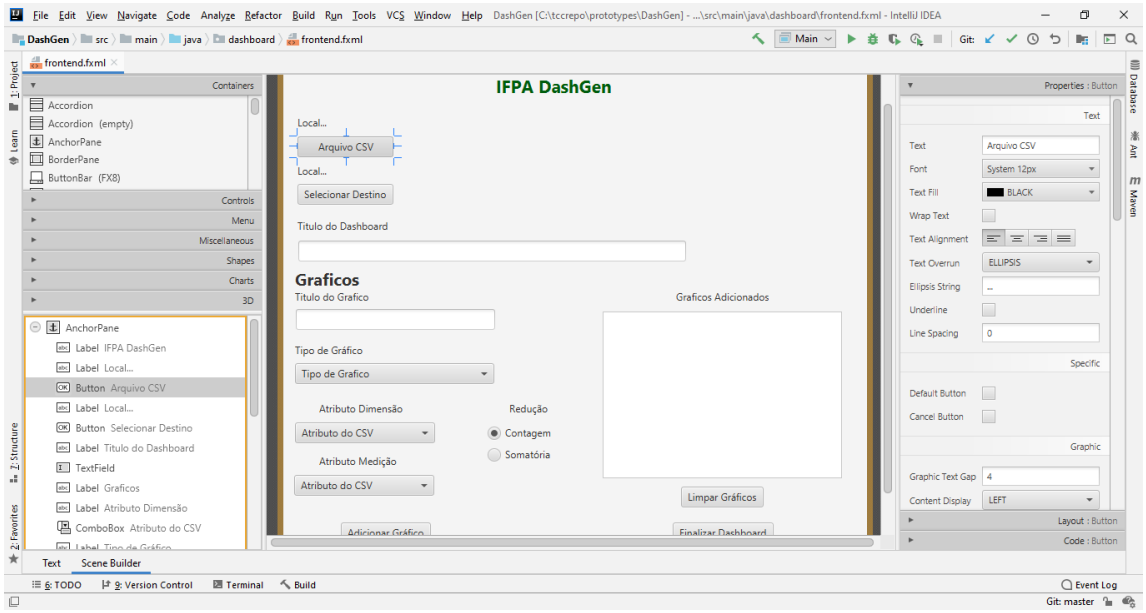
Por ser desenvolvido como uma API Java, uma aplicação JavaFX pode referenciar APIs de qualquer biblioteca Java, como, por exemplo, acessar capacidades nativas do sistema hospedeiro ou conectar-se com aplicações baseadas em servidores (ORACLE, 2019).

As interfaces gráficas em JavaFX são escritas em uma linguagem de marcação baseada no XML chamada FXML. Aliando FXML e CSS, é possível administrar o visual da aplicação sem a necessidade de recompilar o projeto todo, separando totalmente a interface gráfica da lógica de negócios (ORACLE, 2019). A Oracle disponibiliza a ferramenta JavaFX Scene Builder, que é uma IDE WYSIWYG (*what*

you see is what you get, ou “o que você vê é o que obtém”, tradução nossa) com funcionalidades de arrastar e soltar, além de facilidades para parametrização das telas de forma visual, gerando como saída o arquivo FXML. O Scene Builder está disponível como *plugin* em IDEs como o IntelliJ IDEA e o NetBeans (APACHE.ORG, 2019g; JETBRAINS, 2020), facilitando ainda mais o desenho das GUIs sem a necessidade de intercâmbio entre aplicativos durante o desenvolvimento.

Neste trabalho, o IDE utilizado para construir as telas em JavaFX foi o IntelliJ Idea. Na figura 6 encontra-se uma reprodução de tela do IntelliJ Idea com o plugin JavaFX Scene Builder.

Figura 6 – Reprodução de tela do IntelliJ Idea com o Scene Builder



Fonte: Elaborada pelo Autor

Por fim, o JavaFX 8 dá suporte a gráficos em 2D e 3D, suporte a telas de toque e gestos (multitoque), API de manipulação de imagens, codificação de áudio e vídeo, e componente de visualização web, com suporte a Javascript (WIKIPEDIA.ORG, 2019c). É também muito utilizado para desenvolvimento de jogos em Java.

3 SOLUÇÕES CORRELATAS

Este capítulo faz uma breve apresentação de soluções disponíveis no mercado, que permitem a geração de *dashboards*. Tais ferramentas contribuíram para inspirar e identificar funcionalidades para a elaboração deste trabalho de conclusão de curso.

3.1 Qlik Sense

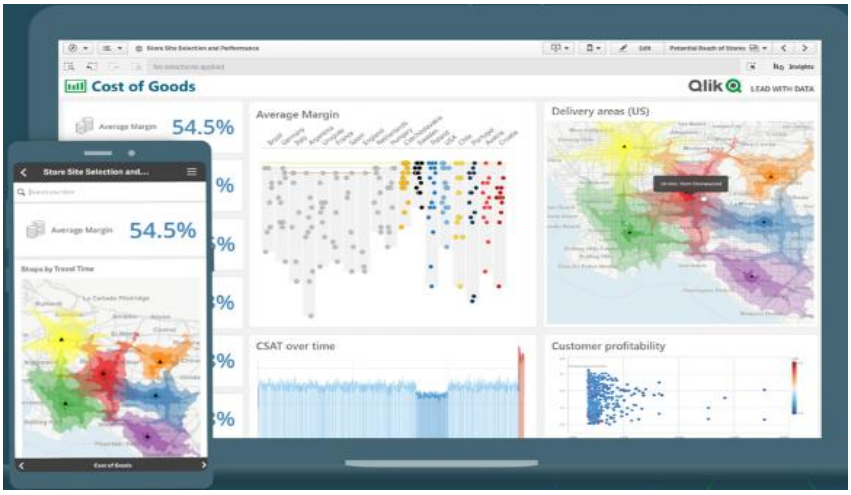
O Qlik Sense é uma ferramenta de visualização de informações que faz uma exploração profunda dos dados, revelando padrões e relações de forma simples e instantânea, entregando informações e conhecimento de negócio (ACADEMIAIN, 2019). Pode ser alimentado por diversas fontes de dados para gerar *dashboards* intuitivos. Possui uma versão gratuita que permite uma análise bastante rica de conjuntos de dados(QLIK, 2019).

Traz como benefícios relevantes (ACADEMIAIN, 2019):

a) Visualização de dados intuitiva: conjunto de gráficos e tabelas que permitem melhor entendimento das relações entre os dados;
b) Facilidade de compartilhamento: Permite fácil compartilhamento de visualizações úteis com outros interessados;
c) Geração de relatórios simplificada: Permite a criação de relatórios personalizados com filtros diferenciados;
d) Exploração associativa e busca inteligente: Possibilita a análise dos dados e criação de verificações mesmo sem qualquer conhecimento em programação, com uso de termos de linguagem natural por exploração associativa;
e) Mobilidade: Com a versão completa é possível usar a plataforma de nuvem, garantindo o acesso em desktops e dispositivos móveis.

Na Figura 7 há uma representação das telas do Qlik Sense.

Figura 7 – Telas do Qlik Sense



Fonte: (QLIK, 2019)

3.2 Tableau Desktop

O Tableau Desktop é um software de análise e visualização de dados. Permite com facilidade conectar-se a diferentes tipos de fontes de dados e possui uma interface com recursos “arrastar-e-soltar” gerando painéis de visualização interativos em poucos segundos sem qualquer conhecimento em programação (TABLEAU, 2019).

Traz como recursos mais relevantes (SOMATIVA, 2019):

a) Análise visual: Poderosas ferramentas de análise permitem filtragem dinâmica de dados, categorização de tendências e análise de coorte aprofunda com uma interface gráfica simples e intuitiva;

b) Gerenciamento de Metadados: Permite a divisão dos dados em diversos campos, combinação de campos em grupos, renomeação de campos e aplicação de campos agrupados como filtro em outras exibições de dados.

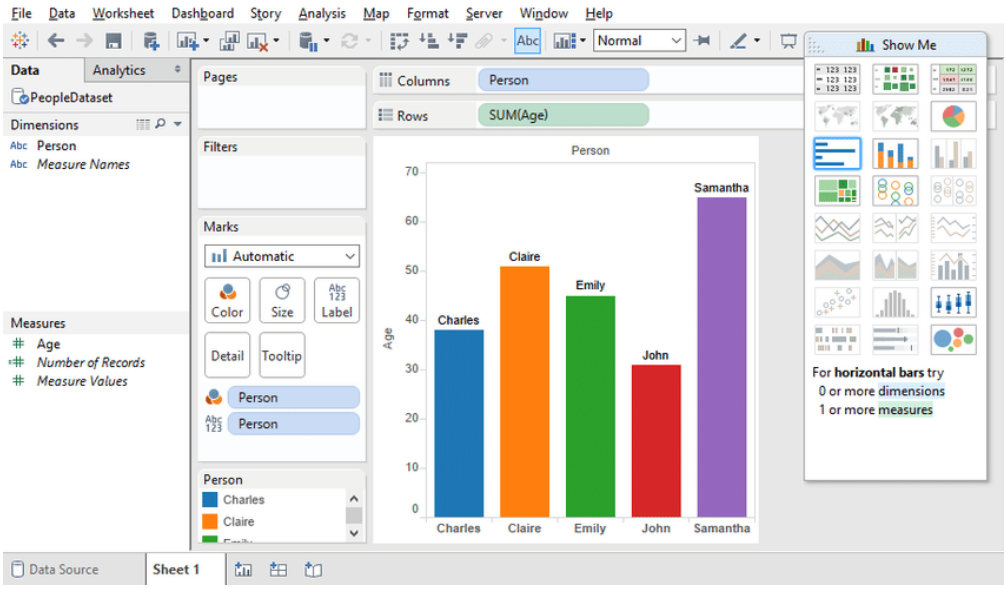
c) Melhores práticas integradas: Inteligência embarcada de anos de pesquisa em análise e visualização de dados permitem a geração automática da melhor disposição gráfica para seus dados.

d) Processador de dados: No caso de necessidade de manter temporariamente os dados off-line ou armazená-los em memória, o processador de dados Tableau permite a extração de volumes de dados para análises Ad hoc.

e) Mapas: Permite o uso de mapas para exibição de dados geográficos, permitindo análise detalhada de informações associadas a localização.

O Tableau conta com plataforma em nuvem para otimizar o processamento de informações e garantir a mobilidade, com aplicativos clientes para *desktops* e dispositivos móveis. Na Figura 8 pode-se observar uma reprodução da tela do Tableau Desktop.

Figura 8 – reprodução de tela do Tableau Desktop



Fonte: (TABLEAU, 2019)

3.3 Microsoft Power BI

O Microsoft Power BI Desktop é um aplicativo para computadores com Windows 10 que permite conexão a fonte de dados para visualização e análise das informações (MICROSOFT, 2019). Com o Power BI Desktop é possível se conectar a uma ou mais fontes de dados e combiná-las em um modelo de dados. Com isso é possível criar visualizações ou conjuntos de visualizações para compartilhamento como relatórios de informação.

Os usos mais comuns da ferramenta Microsoft Power BI são:

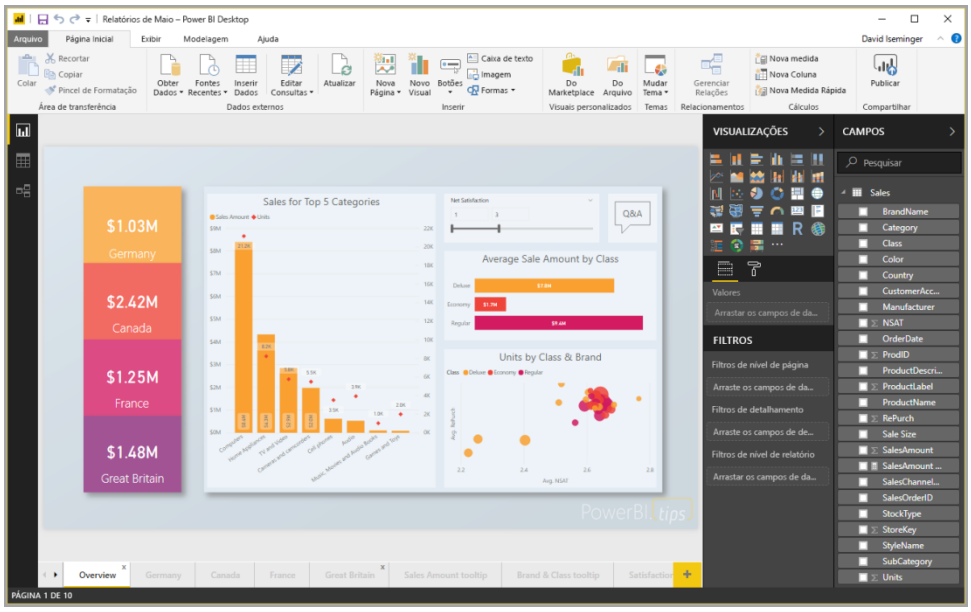
a) Conexão à fontes de dados;

b) Transformação e limpeza dos dados para criação de um modelo de dados;

c) Criação de quadros de visualização para representação visual dos dados;
d) Criação de relatórios;
e) Compartilhamento dos documentos gerados com os demais interessados.

A Microsoft Power BI garante a conexão com os principais bancos de dados disponíveis e oferece conectividade com a plataforma proprietária de nuvem Power BI para aumento da capacidade de processamento de dados e mobilidade, permitindo o acesso a partir de equipamentos desktop e plataformas móveis (MICROSOFT, 2019). Na Figura 9 há uma reprodução de tela da aplicação Microsoft Power BI Desktop.

Figura 9 – Reprodução de tela do Microsoft Power BI Desktop



Fonte: (MICROSOFT, 2019)

4 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo é apresentado o desenvolvimento da ferramenta de geração de *dashboards*, aqui chamada DashGen, utilizando práticas de análise orientada a objetos e das ferramentas citadas no capítulo anterior. Também é demonstrado um exemplo da aplicação objetivando a validação dos resultados.

4.1 Descrição Geral

Partindo de uma especificação de um arquivo de intercâmbio de dados em formato CSV, a aplicação **DashGen** permite gerar um *dashboard* contendo uma quantidade de gráficos desejada pelo usuário. O modelo de saída do gabarito permite um arranjo de três gráficos por linha, dentre três tipos de gráficos com filtros dinâmicos que permitem seleção das dimensões pelo usuário, assim como redefinição para a visão inicial:

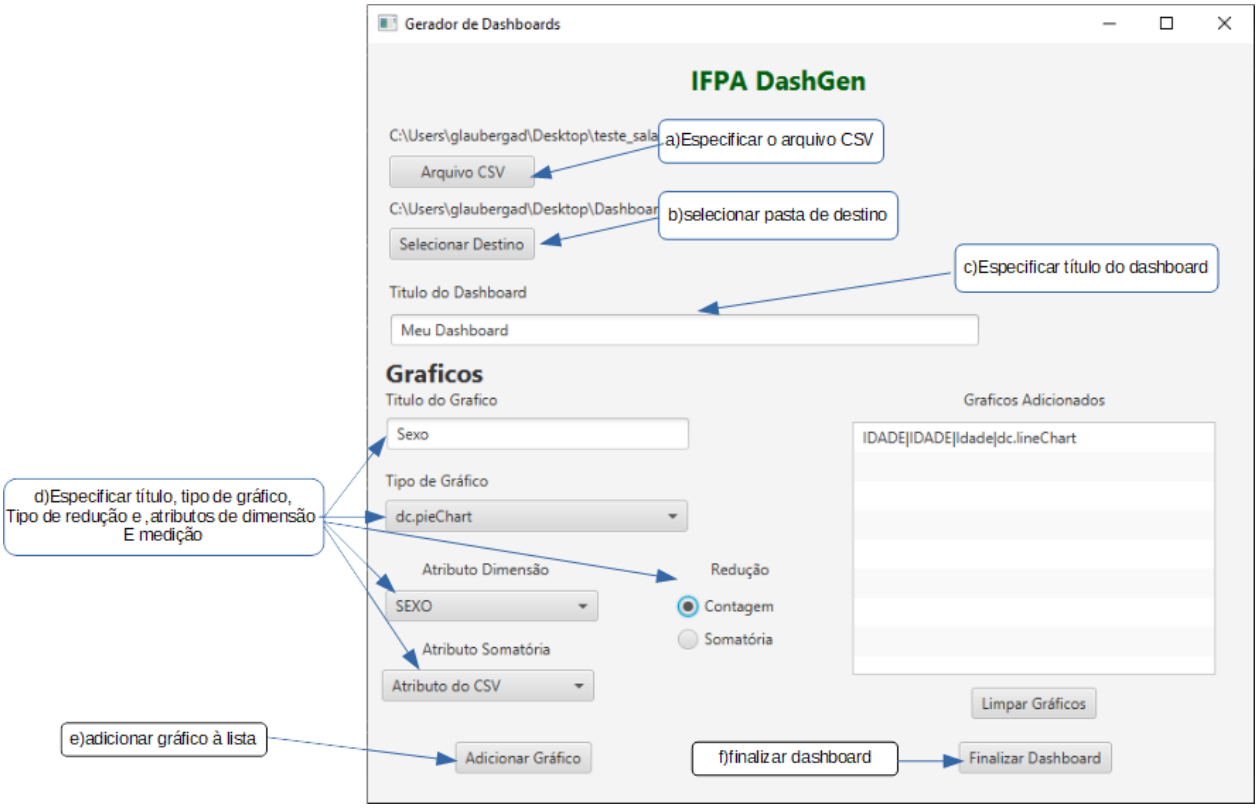
a) gráfico tipo pizza ou torta (*pie*),
b) gráfico de barras horizontais (*row*), e
c) gráfico de linha (*line)*)

O funcionamento da aplicação se dá como segue:

a) O usuário indica ao DashGen qual o arquivo fonte de dados;
b) O usuário determina o diretório onde deve ser armazenado o Dashboard;
c) O usuário determina o título do Dashboard
d) O usuário especifica o atributo que servirá como dimensão de redução para cada gráfico, bem como o título que receberá esse gráfico. No caso de selecionar o tipo de redução de somatória, o usuário deve selecionar o atributo de medição, que deve trazer valores de tipos numéricos. A aplicação filtra somente os atributos numéricos para permitir seleção deste atributo;
e) O usuário adiciona o gráfico à lista.
f) O usuário finaliza o Dashboard. O DashGen salva toda a estrutura necessária para a exibição do dashboard, além de um arquivo compactado em formato ZIP na pasta especificada pelo usuário.

Todo o processo é apresentado em passos na reprodução do protótipo de tela contido na Figura 10.

Figura 10 – Passos do Dashgen



Fonte: Elaborada pelo Autor

4.2 Descrição das Etapas de Desenvolvimento

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

a) Levantamento dos requisitos;
b) Especificação do uso do motor de Templates, baseado nos diagramas UML conceitual de classes e de sequencia;
c) Implementação do motor de Templates;
d) Implementação do gerador de Dashboard.

4.3 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, o arquivo de gabarito, os atributos de dimensão, o título do quadro e o caminho para armazenamento do *dashboard* gerado.

Baseado nisso, foi feito o levantamento de requisitos a serem atendidos, fechando o escopo da aplicação. Na Tabela 4, podem ser observados os requisitos funcionais e na Tabela 5, os requisitos não funcionais.

Tabela 4 – Requisitos Funcionais da aplicação

Requisitos Funcionais		
Requisito	**Descrição**	**Caso de Uso**
RF01	O sistema deve ser capaz de 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, no caminho especificado pelo usuário, copiando automaticamente todos os arquivos necessários.	UC01
RF07	O sistema deve compactar o conteúdo completo do Dashboard em um arquivo.	UC01

Fonte: Elaborada pelo Autor

Tabela 5 – Requisitos não funcionais da aplicação

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.
RNF05	A linguagem de gabaritos a ser aplicada será a Apache FreeMarker.
RNF06	O formato de compactação do arquivo de saída será ZIP.

Fonte: Elaborada pelo Autor

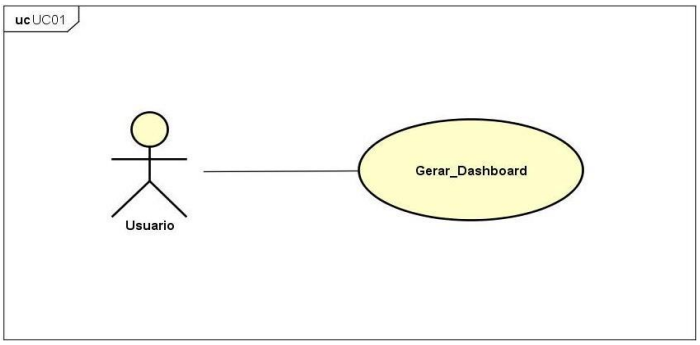
4.4 Modelo de Casos de Uso

O modelo de casos de uso representa as funcionalidades de um sistema observáveis externamente e os elementos externos ao sistema que interagem com

este (BEZERRA, 2015). É um modelo de análise que representa de forma mais refinada os requisitos funcionais do sistema a ser desenvolvido. Este modelo é muito relevante, pois além de direcionar algumas tarefas posteriores ao processo de desenvolvimento, força os desenvolvedores a projetar a solução de acordo com as necessidades específicas do usuário (BEZERRA, 2015).

Na visão do usuário, a aplicação terá um único caso de uso, como pode ser verificado no diagrama de casos de uso UML, representado na Figura 11.

Figura 11 – Diagrama de caso de uso UC01



Fonte: Elaborada pelo Autor

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 na Tabela 6.

Tabela 6 – Descrição do caso de uso UC01

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) O usuário informa o caminho para o arquivo CSV clicando sobre o botão Arquivo CSV; 2) O usuário informa o caminho para armazenamento do dashboard a ser gerado clicando sobre o botão Selecionar Destino; 3) O usuário determina o título do dashboard a ser exibido no topo da página, digitando-o na caixa de texto apropriada; 4) Para inclusão de gráficos, o usuário digita o nome a ser exibido no topo do gráfico na caixa de texto Título do Gráfico; 5) Seleciona o atributo desejado como dimensão principal para o gráfico na lista Atributo de Dimensão; 6) Clica na opção tipo botão de rádio com a redução desejada, tendo como opção a contagem de incidências ou a somatória de um atributo secundário; 7) Se a redução por somatória foi escolhida, o usuário deve selecionar o atributo para soma na lista Atributo Somatória;

	8) O usuário clica sobre o botão Adicionar Gráfico. O gráfico adicionado é exibido na lista Gráficos Adicionados; 9) O usuário repete os passos de 4 a 8 para adicionar quantos gráficos forem desejados; 10) O caso de uso termina no momento em que o usuário clica sobre o botão Finalizar Dashboard. O sistema copia toda a estrutura necessária para o Dashboard para o caminho de destino especificado, além de um arquivo compactado em formato ZIP.
Pós condições:	O DashGen limpa as variáveis de tempo de execução, aguardando a especificação de um novo Dashboard.

Fonte: Elaborada pelo Autor

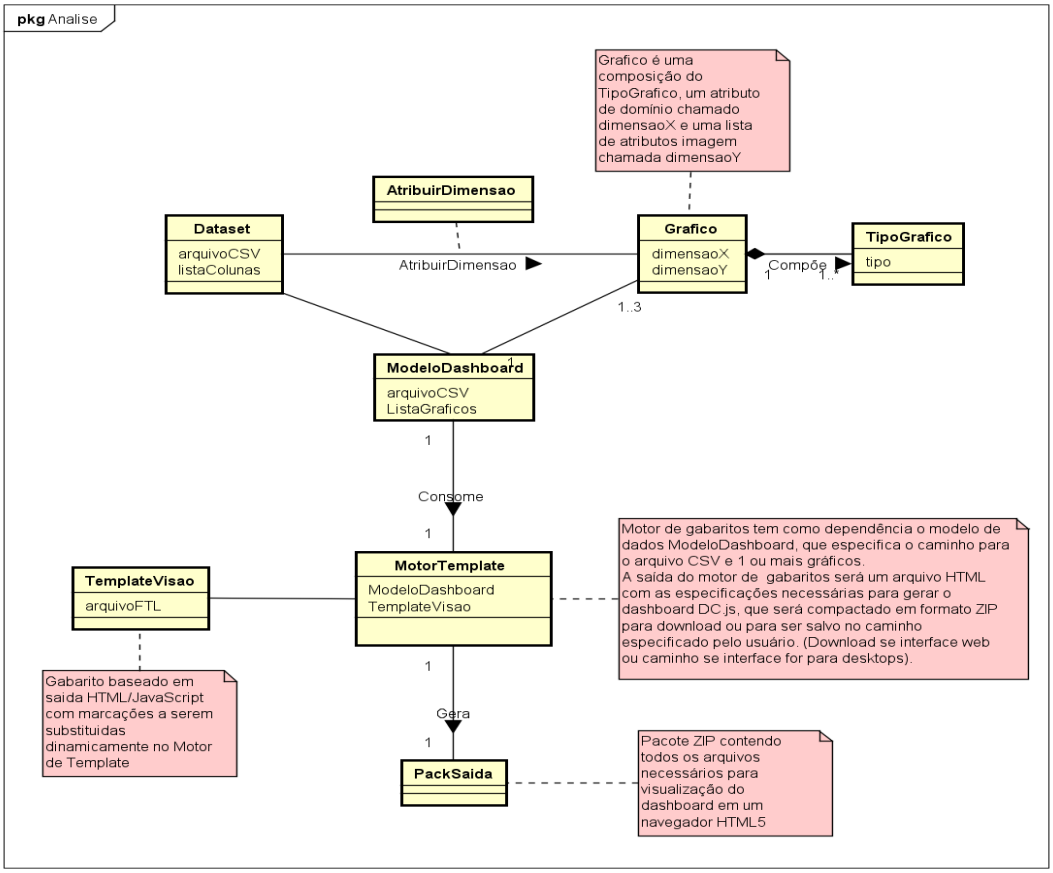
4.5 Modelagem de classes

Durante a maior parte do desenvolvimento da aplicação, o modelo de classes é utilizado e é evoluído a cada iteração. À medida que o sistema vai sendo construído este modelo é incrementado com novos detalhes (BEZERRA, 2015). Dentre os estágios sucessivos de abstração pelos quais passa o modelo de classes, normalmente o inicial é o modelo de classes de análise.

O modelo de classes de análise representa o domínio do problema. Nesta fase ainda não se leva em consideração as restrições da tecnologia a ser empregada na solução (BEZERRA, 2015).

A partir do levantamento de requisitos e da análise do caso de uso UC01, o modelo UML de classes de análise do DashGen foi elaborado como segue representado na Figura 12.

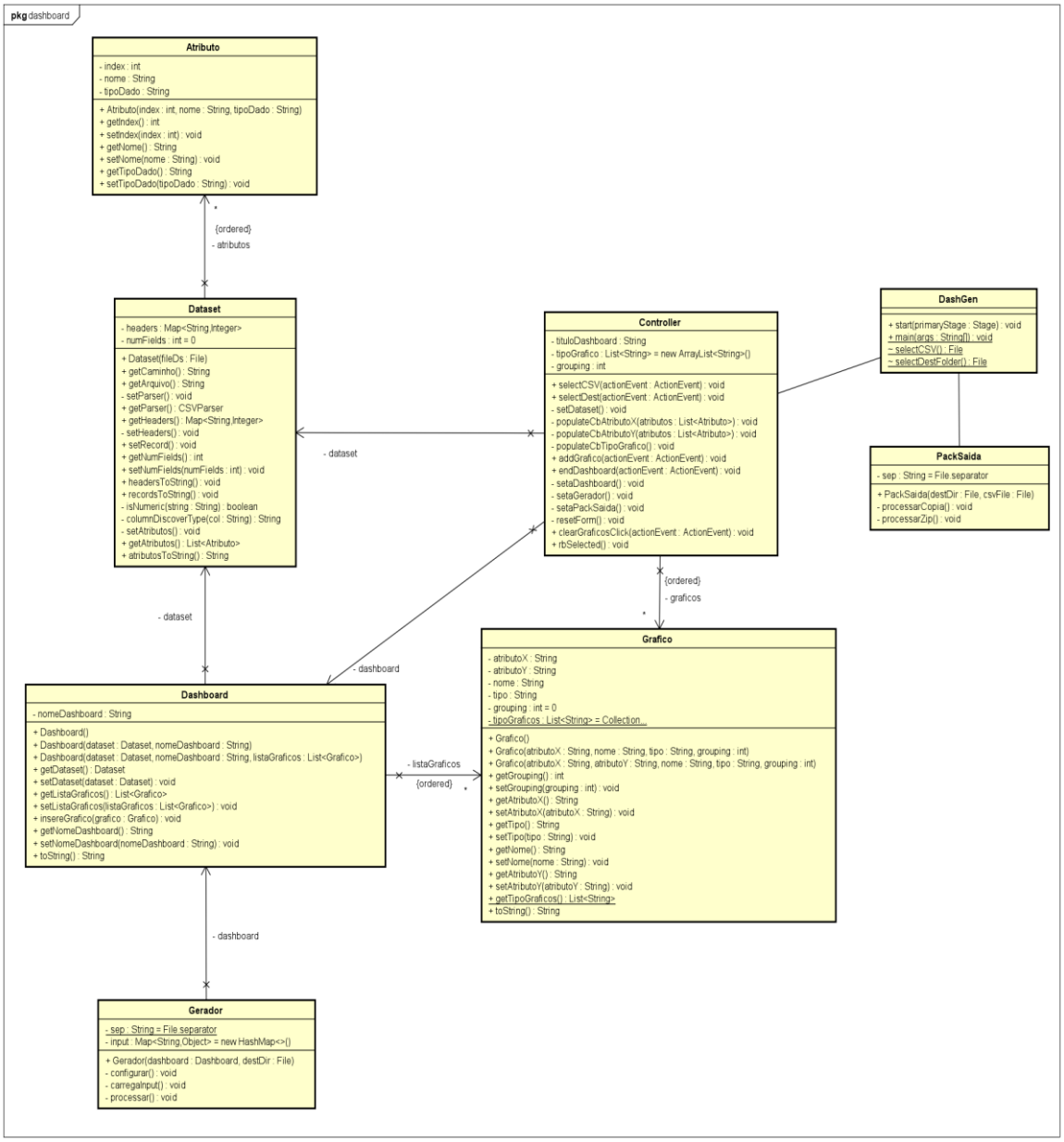
Figura 12 – Diagrama de classes de análise do DashGen



Fonte: Elaborada pelo Autor

De posse do primeiro modelo de classes conceituais (visto na Figura 12), foi iniciado então o processo de análise em nível mais baixo de abstração, com a implementação baseada no Apache Freemarker e na biblioteca DC.js. Após a compreensão na forma de gerar um gráfico simples em uma página HTML5, com o uso dos exemplos disponíveis no site da DC.js (TEAM DC.JS, 2018), ficou claro quais atributos e informações relevantes seriam necessários para a implementação. Com o refinamento obtido neste passo da análise, foi possível desenhar o modelo de classes de implementação, que consiste no modelo de classes na linguagem de programação escolhida (BEZERRA, 2015), neste caso o Java 8. As classes constantes do modelo representado na Figura 13 foram desenhadas importando as classes do código fonte do DashGen para a ferramenta Astah UML 8 (ASTAH, 2019).

Figura 13 – Diagrama de Classes de Implementação DashGen



Fonte: Elaborada pelo Autor

4.6 Modelagem dinâmica

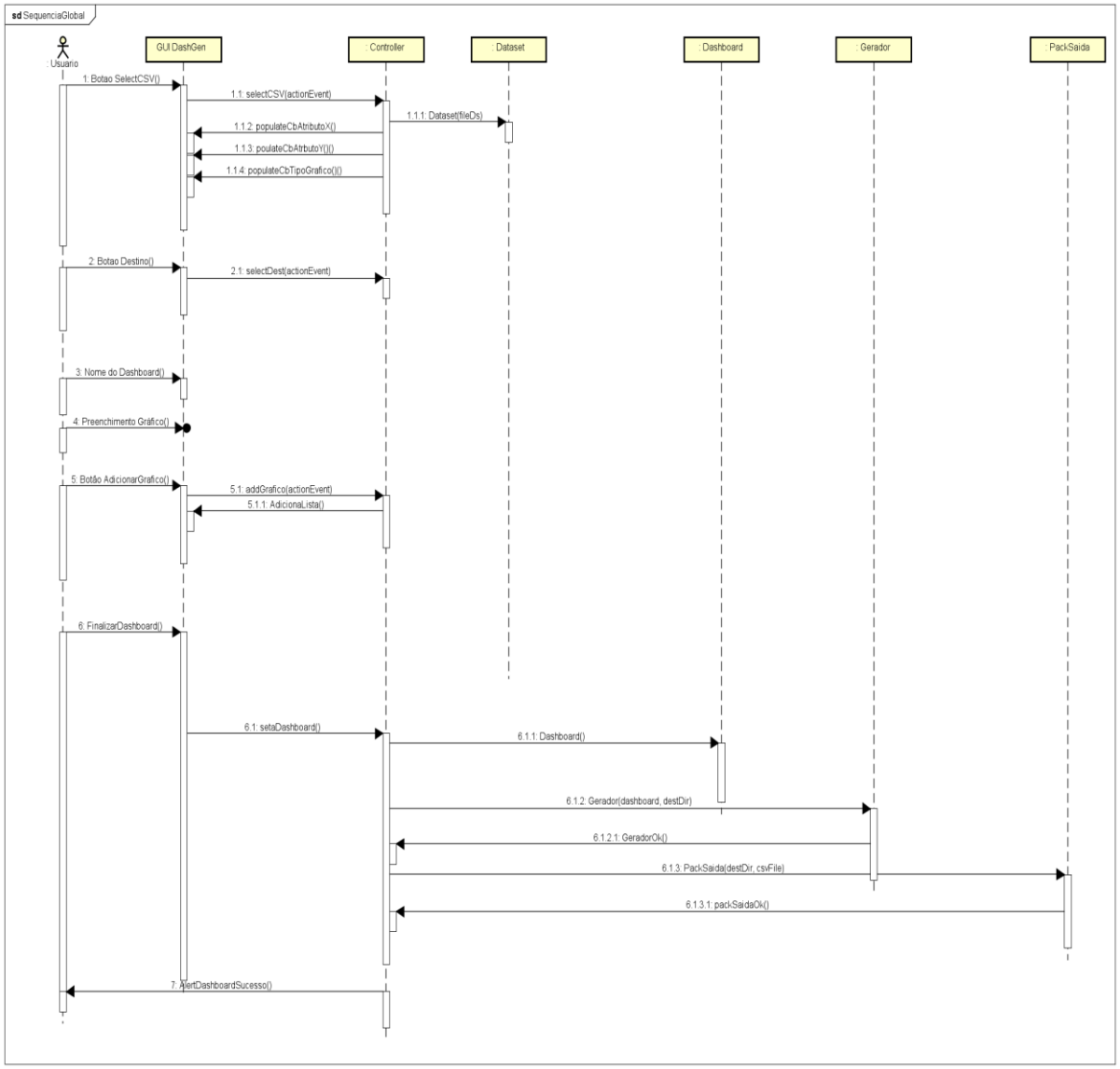
O modelo de interações representa a troca de mensagem entre os objetos para a execução do caso de uso. As etapas anteriores da análise dão uma visão estática e estrutural inicial do sistema. Para se entender a dinâmica do sistema, devemos passar à fase de modelagem dinâmica. Esta modelagem tem o propósito de dar

entendimento do comportamento do sistema em tempo de execução (BEZERRA, 2015).

A ferramenta a ser utilizada nesta etapa é o diagrama de sequência, que tem por finalidade representar a interação entre objetos na sequência temporal em que acontecem (BEZERRA, 2015).

O primeiro diagrama de sequência UML diz respeito a uma visão global das interações das classes do DashGen. A Figura 14 demonstra esta representação.

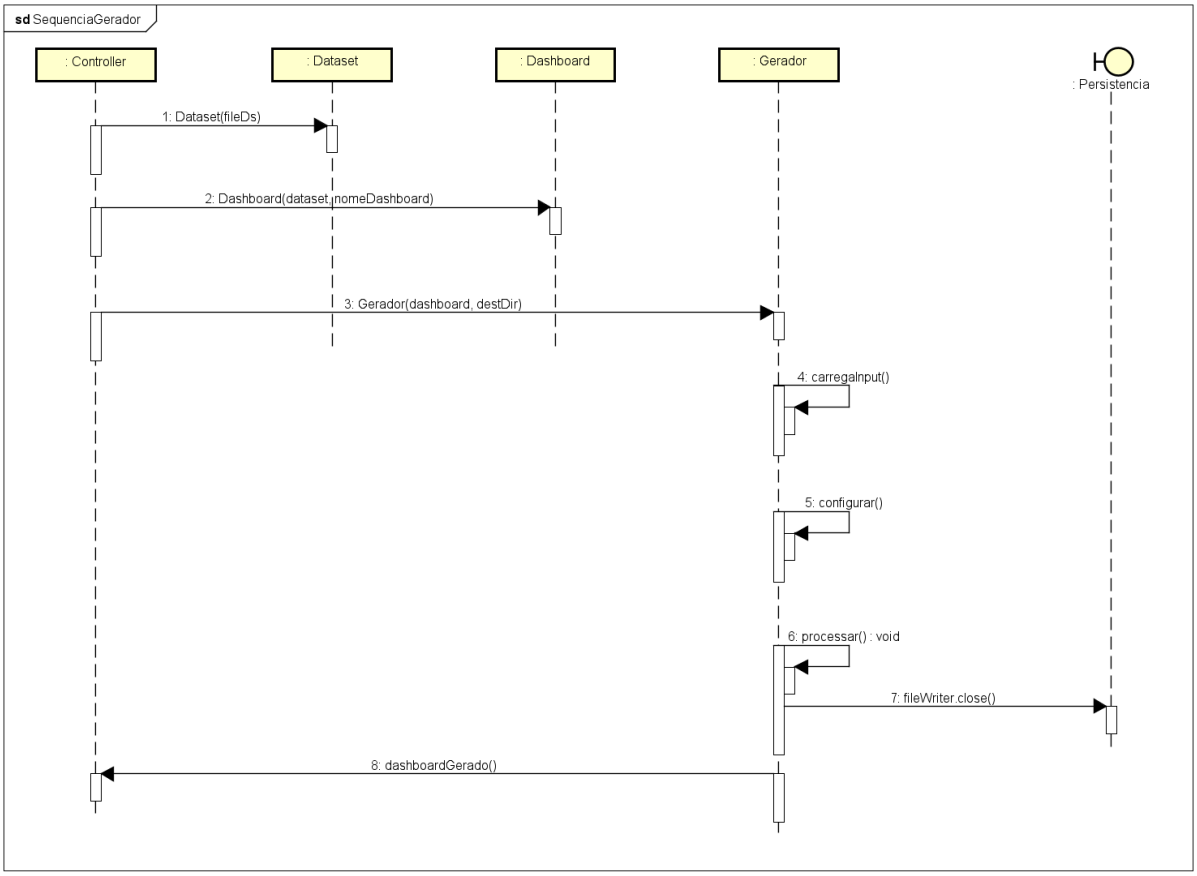
Figura 14 – Diagrama de Sequência Global do DashGen



Fonte: Elaborada pelo Autor

A fim de tornar clara a geração do dashboard de saída, foi também elaborado um diagrama de sequência tratando das interações entre o Controlador principal e a classe <<Gerador>>, responsável pela composição dos dados obtidos em tempo de execução e o gabarito. A Figura 15 contém o diagrama com tal representação.

Figura 15 – Diagrama de sequencia da geração do Dashboard no DashGen



Fonte: Elaborada pelo Autor

4.7 Desenvolvimento da Aplicação DashGen

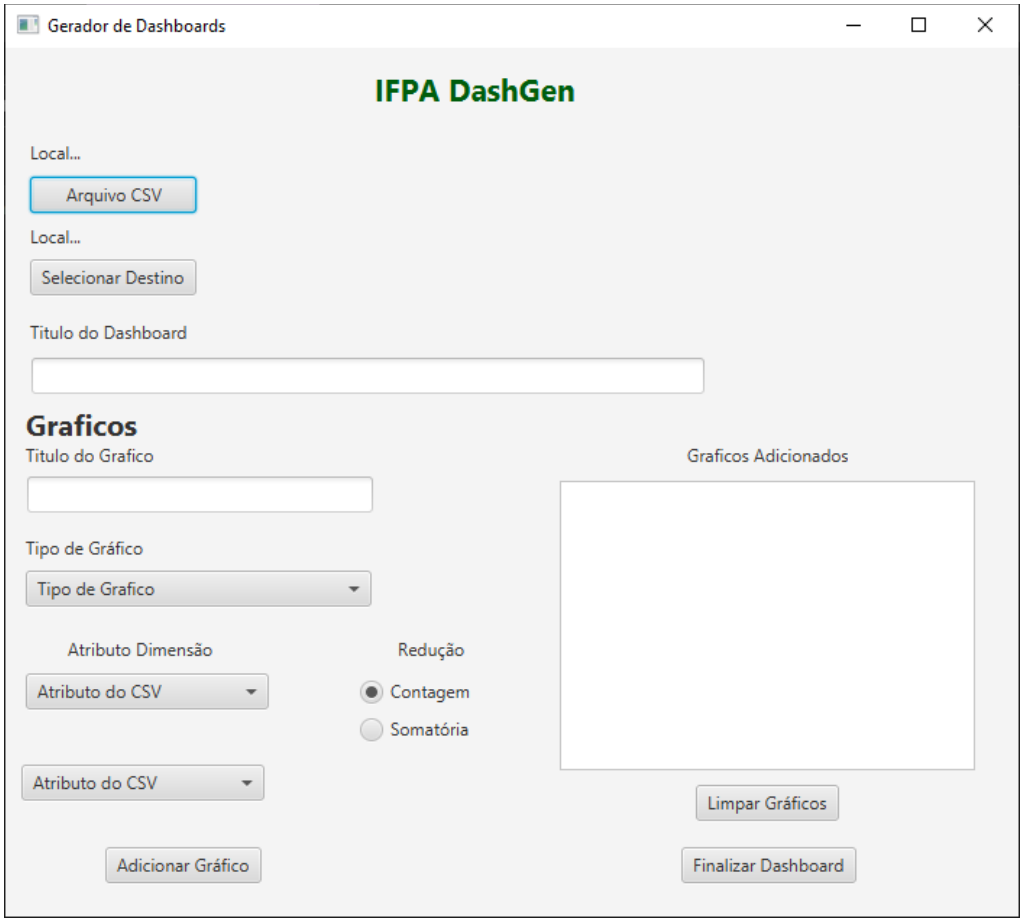
Transpostos os obstáculos das etapas de análise empregadas e descritas nas seções anteriores, foi então iniciado o desenvolvimento da aplicação com GUI JavaFX, com a JDK versão 8. Para tornar mais simples o entendimento, levando em consideração que o padrão arquitetural da aplicação é aderente ao padrão Modelo-Visão-Controle, serão descritas as etapas com a abordagem da camada de Visão em primeiro lugar, partindo em sequencia para a camada de Controle e os Modelos

empregados. Serão descritos de forma mais detalhada os trechos de código fonte mais relevantes.

4.7.1 A interface gráfica do usuário GUI

Considerando a simplicidade da interação do usuário com a aplicação, foi desenvolvida somente uma tela principal contendo todas as informações e campos de entrada necessários para a geração de um dashboard. A tela foi prototipada diretamente no JavaFX Scene Builder (ORACLE, 2019) incorporado ao IntelliJ Idea Ultimate (JETBRAINS, 2020). A Figura 16 mostra uma representação exata da tela em sua versão definitiva.

Figura 16 – Tela do DashGen em sua versão definitiva



Fonte: Elaborada pelo Autor

Na Figura 17 pode-se visualizar o código fonte dos métodos selectCSV() (linha 2), disparado pelo clique sobre o botão “Selecionar CSV”. Esta ação inicia um

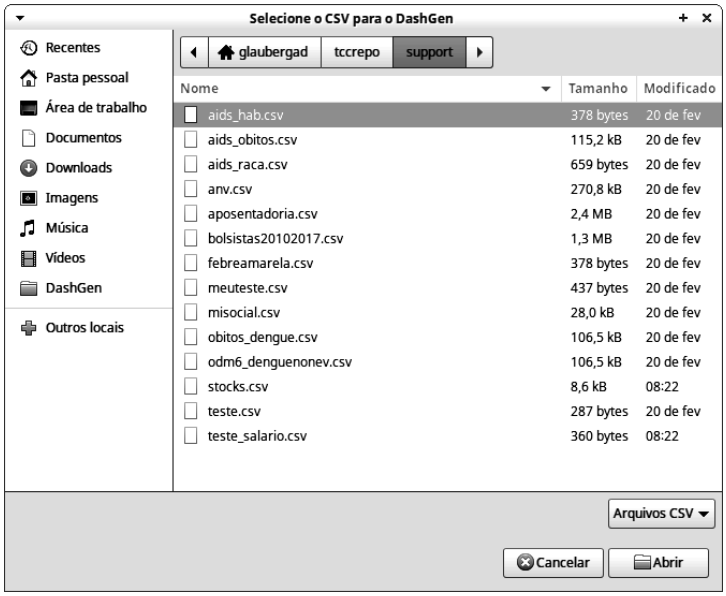
evento que exibe uma janela de seleção de arquivos com filtro pré-definido para *.CSV, como pode ser observado na Figura 18. Também na Figura 17 está o código fonte do método selectDestFolder() (linha 12), disparado pelo clique sobre o botão “Selecionar Destino”. A ação inicia um evento no qual a aplicação exibe uma janela de seleção de diretório no padrão do gerenciador de janelas do sistema hospedeiro, como apresentado na Figura 19. Ambas as janelas de seleção retornam um objeto java.io.File contendo o descritor para o caminho do arquivo ou diretório selecionado.

Figura 17 – Código fonte dos métodos SelectCSV() e SelectDestFolder()

```
01. //JavaFX Filechooser para obter o arquivo CSV. E disparado a partir do PrincipalController.selectCSV()
02.    static File selectCSV()  throws Exception {
03.        FileChooser csvFileChooser = new FileChooser();
04.        csvFileChooser.setTitle("Selecione o CSV para o DashGen");
05.        csvFileChooser.setInitialDirectory(FileSystemView.getFileSystemView().getHomeDirectory());
06.        csvFileChooser.getExtensionFilters().addAll(
07.                new FileChooser.ExtensionFilter("Arquivos CSV", "*.csv"));
08.        return csvFileChooser.showOpenDialog(pStage);
09.    }
10.
11.    //JavaFX DirectoryChooser para obter o diretorio de destino. E disparado a partir do PrincipalController.selectDest()
12.    static File selectDestFolder() throws Exception {
13.        DirectoryChooser destDirectoryChooser = new DirectoryChooser();
14.        destDirectoryChooser.setTitle("Selecione o diretorio para o DashGen");
15.        destDirectoryChooser.setInitialDirectory(FileSystemView.getFileSystemView().getHomeDirectory());
16.        return destDirectoryChooser.showDialog(pStage);
17.    }
```

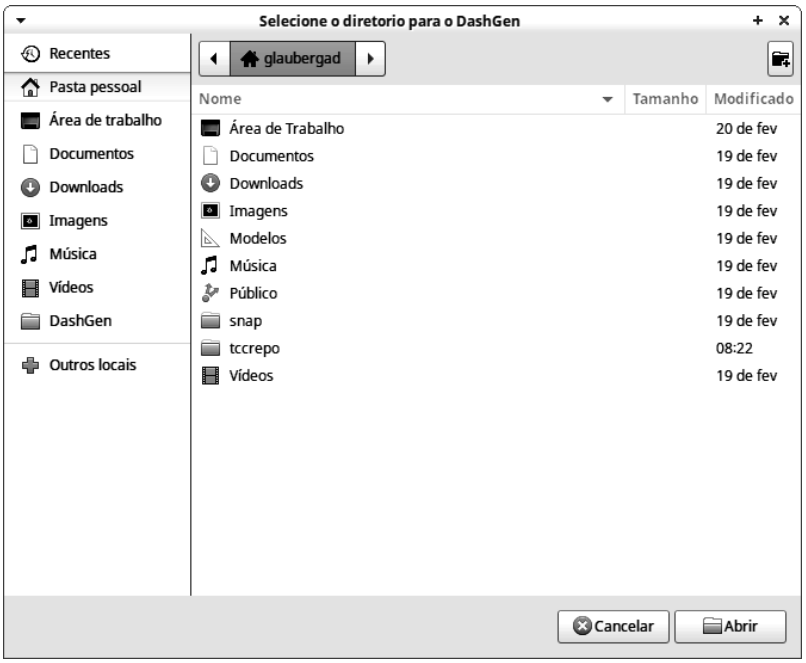
Fonte: Elaborada pelo Autor

Figura 18 – Reprodução da tela de seleção do arquivo CSV



Fonte: Elaborada pelo Autor

Figura 19 – Reprodução da janela de seleção de diretório de destino



Fonte: Elaborada pelo Autor

4.7.2 Controlador

O Controlador é o componente que faz a mediação entre a entrada e a saída, comandando as alterações na visão e no modelo conforme as interações do usuário (WIKIPEDIA.ORG, 2019d). O componente controlador do DashGen é chamado de *Controller*. Ele recebe as ações do usuário, processando as mudanças no modelo e retornando alterações nos componentes visíveis.

Em resumo, alguns métodos desta classe merecem destaque, pela interação direta com os modelos principais. Estes métodos são setDataset(), addGrafico() e endDashboard().

O método setDataset() tem a função de instanciar um objeto baseado na classe de modelo Dataset, passando como parâmetro no construtor o descritor File para o arquivo selecionado pelo usuário. Tem também a funcionalidade de popular as caixas de seleção de atributos e do tipo de gráficos da interface gráfica. A Figura 20 apresenta uma reprodução do código fonte do método.

Figura 20 – Código fonte do método setDataset()

```
01.  //Método instancia um objeto Dataset com o conjunto de dados contido no arquivo CSV selecionado.
02.  private void setDataset() throws Exception {
03.      dataset = new Dataset(csvFile);
04.      populateCbAtributoX(dataset.getAtributos());
05.      if (dataset.getNumFields() > 0) {
06.          populateCbAtributoY(dataset.getAtributos());
07.          lblSomatoria.setVisible(true);
08.          cbAtributoY.setVisible(true);
09.          rbReduceSum.setVisible(true);
10.      } else {
11.          cbAtributoY.setVisible(false);
12.          rbReduceSum.setVisible(false);
13.      }
14.      populateCbTipoGrafico();
15.  }
```

Fonte: Elaborada pelo Autor

O método addGrafico() tem a função importante de preencher uma instância do tipo List com objetos tipo Grafico, que, assim como o objeto Dataset, são vitais para a geração do Dashboard. Este método é disparado no momento em que o usuário clica no botão “Adicionar Gráfico”. Assim, a aplicação adiciona um novo item à lista com os dados preenchidos e os atributos e método de redução selecionados, atualizando a lista de visualização contida na tela. Na Figura 21 demonstra-se um trecho do código fonte do método citado.

Figura 21 – código fonte do método addGrafico()

```
01.  public void addGrafico(ActionEvent actionEvent) {
02.      try {
03.          String atributoX = cbAtributoX.getSelectionModel().getSelectedItem().toString();
04.          String atributoY;
05.          if (this.grouping == 1) {
06.              atributoY = cbAtributoY.getSelectionModel().getSelectedItem().toString();
07.          } else {
08.              atributoY = cbAtributoX.getSelectionModel().getSelectedItem().toString();
09.          }
10.          String tipoGraf = cbTipoGrafico.getSelectionModel().getSelectedItem().toString();
11.          String titGraf = tfTituloGrafico.getText();
12.          graficos.add(new Grafico(atributoX, atributoY, titGraf, tipoGraf, this.grouping));
13.          lvGraficos.getItems().setAll(graficos);
14.          cbAtributoX.getSelectionModel().clearSelection();
15.          cbAtributoY.getSelectionModel().clearSelection();
16.          btnEndDashboard.setDisable(false);
17.      } catch (Exception e) {
18.          System.out.println("AddGrafico:" + e.toString());
19.          new Alert(Alert.AlertType.INFORMATION, "Voce nao selecionou os dados para inserir o grafico!").showAndWait();
20.      }
21.  }
```

Fonte: Elaborada pelo Autor

Na Figura 22 pode-se observar o código fonte do método endDashboard e suas dependências. O método endDashboard() (linha 2) é disparado quando o usuário clica sobre o botão “Finalizar Dashboard” (na interface gráfica), cuja finalidade é fazer a instanciação do objeto Dashboard, passando como parâmetros para o construtor o Dataset, o Nome especificado pelo usuário e a lista de Gráficos

com a chamada de método setaDashboard() (linha 16). Também instancia o objeto Gerador com o método setaGerador() (linha 22), passando como parâmetros para o construtor o objeto Dashboard e o descritor File do diretório de destino. Por fim, instancia o objeto PackSaida com o método setaPackSaída() (linha 27), passando como parâmetro para o construtor o descritor File do diretório de destino e o descritor File do arquivo CSV.

Figura 22 – Código fonte do método endDashboard() e suas dependências

```
01. //Metodo finaliza a geracao do Dashboard, executando o motor de templates FreeMarker, copiando os arquivos para a pasta de destino e gerando o zip de saida.
02. public void endDashboard(ActionEvent actionEvent) {
03.     setaDashboard();
04.     try {
05.         setaGerador();
06.         setaPackSaida();
07.     } catch (Exception e) {
08.         new Alert(Alert.AlertType.INFORMATION, "Nao foi possivel gerar o Dashboard. Verifique as opcoes escolhidas!").showAndWait();
09.     } finally {
10.         new Alert(Alert.AlertType.INFORMATION, "Dashboard gerado com sucesso!").showAndWait();
11.         resetForm();
12.     }
13. }
14.
15. //Metodo instancia objeto Dashboard juntando todas as informacoes definidas pelo usuario na UI
16. private void setaDashboard() {
17.     tituloDashboard = tfTituloDashboard.getText();
18.     dashboard = new Dashboard(dataset, tituloDashboard, graficos);
19. }
20.
21. //Metodo instancia a classe gerador, que processa o template FreeMarker e gera em disco o arquivo Dashboard.html.
22. private void setaGerador() throws IOException, TemplateException {
23.     new Gerador(dashboard,destFolder);
24. }
25.
26. //Metodo instancia novo pack  de saida para gerar em disco o Dashboard completo.
27. private void setaPackSaida() throws IOException {
28.     new PackSaida(destFolder, csvFile);
29. }
```

Fonte: Elaborada pelo Autor

4.7.3 A classe Dataset

A fim de dar a separação devida a cada elemento necessário para a geração de um dashboard completo, foi criada a classe Dataset. Esta tem a função de agregar em um objeto concreto, as informações necessárias para o DashGen relativas ao arquivo CSV.

Esta classe tem duas particularidades bastante relevantes que merecem destaque. Ela implementa a biblioteca Apache CommonsCSV (APACHE.ORG, 2019e) para manipulação dos dados do arquivo CSV, e, ainda, possui um método de identificação por força bruta de atributos contendo dados em formato numérico.

No tocante à implementação da CommonsCSV, foram utilizadas as classes CSVParser e CSVRecord.

A classe CSVParser foi empregada com a finalidade de se obter os cabeçalhos com os rótulos dos atributos contidos no arquivo, com o método getHeaderMap(), que retorna um Map<String,Integer>. Com isso foi possível formar uma lista dos atributos contidos no arquivo, facilitando a seleção pelo usuário.

A classe CSVRecord permite a iteração das registros contidos nas linhas do arquivo CSV. Esta iteração foi necessária para embarcar um método de identificação de atributos com conteúdos de tipo numérico por força bruta. Se o atributo de medição não for composto somente de valores numéricos, é impossível aplicar o método de redução por somatória..

Na Figura 23 visualiza-se a reprodução do código fonte dos métodos isNumeric() (linha 2) e columnDiscoverType() (linha 12), responsáveis por percorrer as 100 primeiras linhas do arquivo em cada um dos atributos testando o resultado e identificando os valores numéricos.

Figura 23 – Código fonte dos métodos isNumeric() e columnDiscoverType()

```
01. //Metodo para teste se uma String contem um valor numerico
02.   private boolean isNumeric(String string) {
03.     try {
04.         Double.parseDouble(string);
05.         return true;
06.     } catch (NumberFormatException e) {
07.         return false;
08.     }
09. }
10.
11. //metodo que testa as 100 primeiras linhas do CSV a procura de valores numericos
12. private String columnDiscoverType(String col) {
13.     int numeric = 0, string = 0;
14.     //Necessario resetar o Record para ler novamente as linhas do CSV a partir do inicio
15.     this.setRecord();
16.     for (CSVRecord reg : this.record) {
17.         if (this.isNumeric(reg.get(col))) {
18.             numeric++;
19.         } else {
20.             string++;
21.         }
22.         if (reg.getRecordNumber() == 100)
23.             break;
24.     }
25.     if (string > 0) {
26.         return "T";
27.     } else {
28.         this.numFields++;
29.         return "N";
30.     }
31. }
```

Fonte: Elaborada pelo Autor

4.7.4 A classe Gerador e o gabarito dashboard.ftl

A classe denominada Gerador é a responsável por operar o Motor de Gabaritos Apache Freemarker. Deve receber como parâmetro de construtor o objeto

Dashboard e o descritor File do diretório de destino especificado pelo usuário. Com o intuito de tornar o código mais legível e facilitar o entendimento de cada etapa da configuração do Freemarker, foram criados os métodos configurar(), carregaInput() e processar(), representados na Figura 24 .

O método configurar() (linha 2) tem a finalidade de atribuir todas as informações mínimas requeridas pelo objeto cfg, instância da classe freemarker.template.Configuration. Dentre essas informações estão o caminho absoluto para carga dos gabaritos. Neste também é atribuído o gabarito ao objeto template, instância da classe freemarker.template.Template.

O método carregaInput() (linha 13) é responsável por atribuir todos os dados vindos do objeto Dashboard a um Hashmap<String,Object>. Esta coleção corresponde à lista de parâmetros a serem passados para o Motor Freemarker fazer a composição dos dados estáticos do gabarito com os dados dinâmicos nas marcações.

O método processar() (linha 21) tem como funcionalidade criar o arquivo final no diretório de destino, tendo como conteúdo a saída do método process() do objeto template, que recebe como parâmetros o Hashmap<String,Object> input, que foi alimentado no método carrgaInput() e o descritor para o arquivo de saída.

Figura 24 – Código fonte dos métodos da classe Gerador

```
01. //Configurador dos dados necessarios para processamento do template
02.  private void configurar() throws IOException {
03.      cfg.setDirectoryForTemplateLoading(HOME);
04.      cfg.setIncompatibleImprovements(new Version(2, 3, 28));
05.      cfg.setDefaultEncoding("UTF-8");
06.      cfg.setLocale(new Locale("pt","BR"));
07.      cfg.setTemplateExceptionHandler(TemplateExceptionHandler.RETHROW_HANDLER);
08.      this.template = cfg.getTemplate("dashboard.ftl");
09.  }
10.
11.
12.  //Metodo responsavel por criar um HashMap com todos os dados e objetos a serem preenchidos sobre o template
13.  private void carregaInput(){
14.      input.put("arquivo", dashboard.getDataset().getArquivo());
15.      input.put("titulo", dashboard.getNomeDashboard());
16.      input.put("graficos", dashboard.getListaGraficos());
17.  }
18.
19.  //Executa o processamento do template e a geracao do arquivo final, gravando no caminho especificado sob o nome
20.  // dashboard.html
21.  private void processar() throws IOException, TemplateException {
22.      BufferedWriter fileWriter = new BufferedWriter(new OutputStreamWriter(
23.  FileOutputStream(new File(destDir.getAbsolutePath()  +sep+"dashboard.html")),"UTF-8"));
24.      template.process(input, fileWriter);
25.      fileWriter.close();
26.  }
```

Fonte: Elaborada pelo Autor

O arquivo de gabarito *dashboard.ftl* contém o código que se manterá estático no conteúdo do arquivo final do dashboard, entremeado com marcações usando as convenções da Freemarker Template Language que serão substituídas em tempo de execução por dados que serão submetidos a comparações lógicas e iteradores de listas ou coleções ou simplesmente escritos. Na Figura 25 visualiza-se o conteúdo completo do gabarito.

Figura 25 – Conteúdo fonte do arquivo *dashboard.ftl*

```
1.    <!doctype html>
2.  <html lang="pt-br">
3.  <head>
4.      <title>${titulo}</title>
5.      <!-- Required meta tags -->
6.      <meta charset="utf-8">
7.      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8.      <!-- Bootstrap e DC CSS -->
9.      <link rel="stylesheet" href="css/bootstrap.min.css">
10.     <link rel="stylesheet" href="css/dc.css">
11.     <link rel="stylesheet" href="css/dashgen.css">
12.     <script src=js/crossfilter.js></script>
13.     <script src=js/d3.js></script>
14.     <script src=js/dc.js></script>
15. </head>
16. <body>
17. <div class="container">
18.     <div class="row dashgen-header" >
19.         <img src="img/ifpa_navlogo.png" width="120px" height="120px" alt="Logo IFPA"/>
20.         <h1>${titulo}</h1>
21.     </div>
22.     <div class="row">
23.         <#list graficos as grafico>
24.             <div id="grafico${grafico_index +1}" class='col-xs-12 col-md-12 col-lg-4'>
25.             <strong>${grafico.nome}</strong>
26.             <span class="reset" style="display: none;">Selecionado:<span class="filter">
27.         </span></span>
28.
    <a class="reset" href="javascript:grafico${grafico_index +1}.filterAll();dc.redrawAll();"style=
    "display: none;">Limpar</a>
29.                 <div class="clearfix"></div>
30.             </div>
31.         </#list>
32.     </div>
33.     <div class="row dashgen-footer text right"> Gerado Automaticamente pelo Dashgen. IPFA 2020
    </div>
34. </div>
35. <!-- necessário para o bootstrap -->
36. <script src="js/jquery-3.4.1.min.js"></script>
37. <!-- Aqui começa a  geração dos gráficos -->
38. <script>
39.
40.     //instanciacao e identificacao do local de exibicao no DOM
41.     <#list graficos as grafico>
42.     var grafico${grafico_index+1} = ${grafico.tipo}("#grafico${grafico_index+1}");
43.     </#list>
44.
45.     //Especificacao da fonte de dados
46.     var url = "data/${arquivo}";
```

```
47.
48.     d3.csv(url, function (err, data) {
49.         if (err) throw err;
50.         var ndx = crossfilter(data);
51.
52.         //Estabelecimento da dimensao principal do grafico
53.         <#list graficos as grafico>
54.         var grafico${grafico_index+1}Dim = ndx.dimension(d => d.${grafico.atributoX});
55.         </#list>
56.
57.         //Redutores
58.         <#list graficos as grafico>
59.         <#if grafico.grouping == 1>
60.         var grafico${grafico_index+1}Group = grafico${grafico_index+1}Dim.group().reduceSum(d =
    > d.${grafico.atributoY});
61.         var grafico${grafico_index+1}minX = grafico${grafico_index+1}Dim.bottom(1)[0]["${grafic
    o.atributoY}"];
62.         var grafico${grafico_index+1}maxX = grafico${grafico_index+1}Group.top(1).value;
63.         <#else>
64.         var grafico${grafico_index+1}Group = grafico${grafico_index+1}Dim.group();
65.         var grafico${grafico_index+1}minX = grafico${grafico_index+1}Dim.bottom(1)[0]["${grafic
    o.atributoX}"];
66.         var grafico${grafico_index+1}maxX = grafico${grafico_index+1}Dim.top(1)[0]["${grafico.a
    tributoX}"];
67.         </#if>
68.         </#list>
69.
70.         //Parametros especificos de cada grafico
71.         <#list graficos as grafico>
72.         grafico${grafico_index+1}
73.         <#if grafico.tipo != "dc.lineChart">
74.             //Exibe os cinco atributos com valores mais altos, agrupando os demais em Others
75.             .cap(5)
76.             </#if>
77.             .dimension(grafico${grafico_index+1}Dim)
78.             .group(grafico${grafico_index+1}Group)
79.             <#if grafico.tipo == "dc.lineChart">
80.             //Define a escala do eixo X baseado nos valores minimo e maximo do agrupamento de r
    educao.
81.             .x(d3.scale.linear().domain([grafico${grafico_index+1}minX, grafico${grafico_index+
    1}maxX]))
82.             </#if>
83.             <#if grafico.tipo == "dc.rowChart">
84.             //Garante o reajuste automatico do eixo X
85.             .elasticX(true)
86.         </#if>
87.         ;
88.         </#list>
89.         dc.renderAll();
90.     });
91. </script>
92. </body>
93. </html>
```

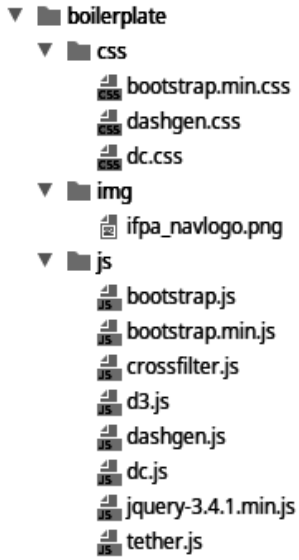
Fonte: Elaborada pelo Autor

4.7.5 A classe PackSaida

Após a execução dos métodos da classe Gerador, é necessário fazer a composição do diretório de destino, pois o arquivo dashboard.html tem dependências CSS e JavaScript além das bibliotecas DC.js, D3.js e Crossfilter. A

solução para essa necessidade foi criar no diretório principal da aplicação um diretório chamado *boilerplate*. A Figura 26 mostra a estrutura deste diretório.

Figura 26 – O conteúdo do diretório *boilerplate*



Fonte: Elaborada pelo Autor

A Classe PackSaida tem duas funções bastante relevantes. A primeira é copiar o arquivo CSV e toda a estrutura do diretório *boilerplate* para o diretório de destino especificado pelo usuário e a segunda função é criar um arquivo chamado *dashgen.zip*, que consiste no conteúdo do dashboard e suas dependências compactados em formato ZIP. Na figura 27 , visualiza-se uma reprodução do código fonte dos métodos processarCopia() e processarZip().

O método processarCopia() (linha 2) se beneficia do reúso da biblioteca Apache Commons IO FileUtils (APACHE.ORG, 2019f). O método copyFile() faz a cópia do arquivo CSV para um diretório denominado *data* no caminho especificado para o diretório destino. O método copyDirectory() copia o conteúdo do diretório *boilerplate* recursivamente para o caminho especificado.

O método processarZip() (linha 7) utiliza a biblioteca zeroturnaround.zip.ZipUtil (RAUDJÄRV, 2019) para compactar recursivamente o conteúdo do diretório de destino com todos os arquivos em seu estado final, gerando um arquivo de saída nomeado como dashgen.zip no diretório de destino especificado pelo usuário.

Figura 27 – código fonte dos métodos processarCopia() e ProcessarZip()

```
01. //Usa FileUtils para facilitar o processo de copia recursiva do diretorio e subdiretorios
02. private void processarCopia() throws IOException {
03.     FileUtils.copyFile(this.csvFile, new File(this.destDir.getAbsolutePath() + sep + "data" + sep + csvFile.getName()));
04.     FileUtils.copyDirectory(this.boilerplateDir, this.destDir);
05. }
06. //processa a compactacao dos arquivos contidos no diretorio de origem da aplicacao em um arquivo ZIP no diretorio de destino
07. private void processarZip() {
08.     String FILE_NAME = "dashgen.zip";
09.     ZipUtil.pack(this.destDir, new File(this.destDir.getAbsolutePath() + sep + FILE_NAME));
10. }
```

Fonte: Elaborada pelo Autor

4.8 Validação da Aplicação DashGen

Para validação da eficácia da aplicação DashGen na geração de dashboards foi aplicado um estudo de caso com um arquivo em formato CSV com dados fictícios de entrevistados das cinco regiões do Brasil. O arquivo denominado teste_salario.csv tem como atributos o sexo do entrevistado, a idade, a região e o salário em Reais. Na Tabela 7 podemos visualizar as dez primeiras linhas do arquivo.

Tabela 7 – primeiras linhas do arquivo teste_salario.csv

SEXO	IDADE	REGIAO	SALARIO
M	23	NORTE	1200.00
M	23	NORTE	1550.00
F	25	SUL	1100.00
M	21	SUDESTE	1120.00
F	33	CENTRO-OESTE	1000.00
F	27	NORDESTE	1110.00
F	33	SUDESTE	1000.00
F	42	SUDESTE	998.00
M	40	NORDESTE	1897.00
M	41	NORDESTE	1990.00

Fonte: Elaborada pelo Autor

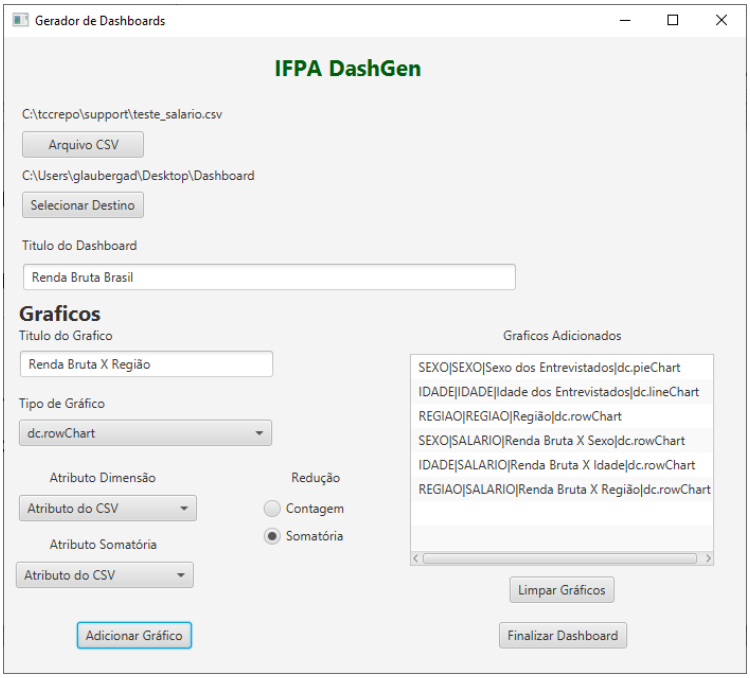
Considerando os atributos contidos no arquivo, para esta simulação, foram definidos os seguintes requisitos:

a) Arquivo CSV: “c:\tccrepo\support\teste_salario.csv”.
b) Pasta de destino: ”%USERPROFILE%\desktop\dashboard”
c) Título do dashboard: “Renda Bruta Brasil”.

d) Um gráfico tipo pizza com o atributo de dimensão SEXO - Redução por Contagem – Rótulo: “Sexo dos Entrevistados”.
e) Um gráfico tipo linha com o atributo de dimensão IDADE – Redução por Contagem – Rótulo: “Idade dos Entrevistados”.
f) Um gráfico tipo barras horizontais com o atributo de dimensão REGIAO – Redução por Contagem – Rótulo: “Região”.
g) Um gráfico tipo barras horizontais com o atributo de dimensão SEXO – Redução por Somatória – Atributo de medição SALARIO – Rótulo: “Renda Bruta X Sexo”.
h) Um gráfico tipo barras horizontais com o atributo de dimensão IDADE – Redução por Somatória – Atributo de medição SALARIO – Rótulo: “Renda Bruta X Idade”.
i) Um gráfico tipo barras horizontais com o atributo de dimensão REGIAO – Redução por Somatória – Atributo de medição SALARIO – Rótulo: “Renda Bruta X Região”.

Na execução da aplicação DashGen, a interface gráfica com os dados inseridos em conformidade com os valores propostos pode ser vista na Figura 28.

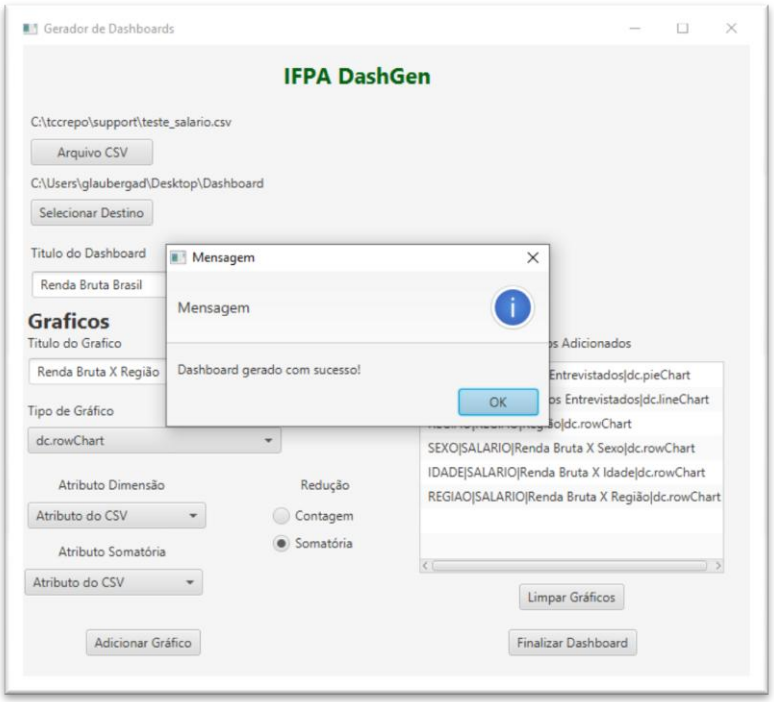
Figura 28 – Instância do DashGen com dados inseridos



Fonte: Elaborada pelo Autor

Ao clicar sobre o botão “Finalizar Dashboard”, o DashGen processa o gabarito e o modelo de dados do dashboard, produzindo o artefato final *dashboard.html* e copia as dependências contidas no diretório de origem *boilerplate* para o diretório de destino. Também neste mesmo diretório, a aplicação gera o arquivo compactado dashgen.zip. Ao término deste processamento, a interface gráfica retorna um painel de alerta confirmando o sucesso na geração do dashboard, conforme se visualiza na Figura 29.

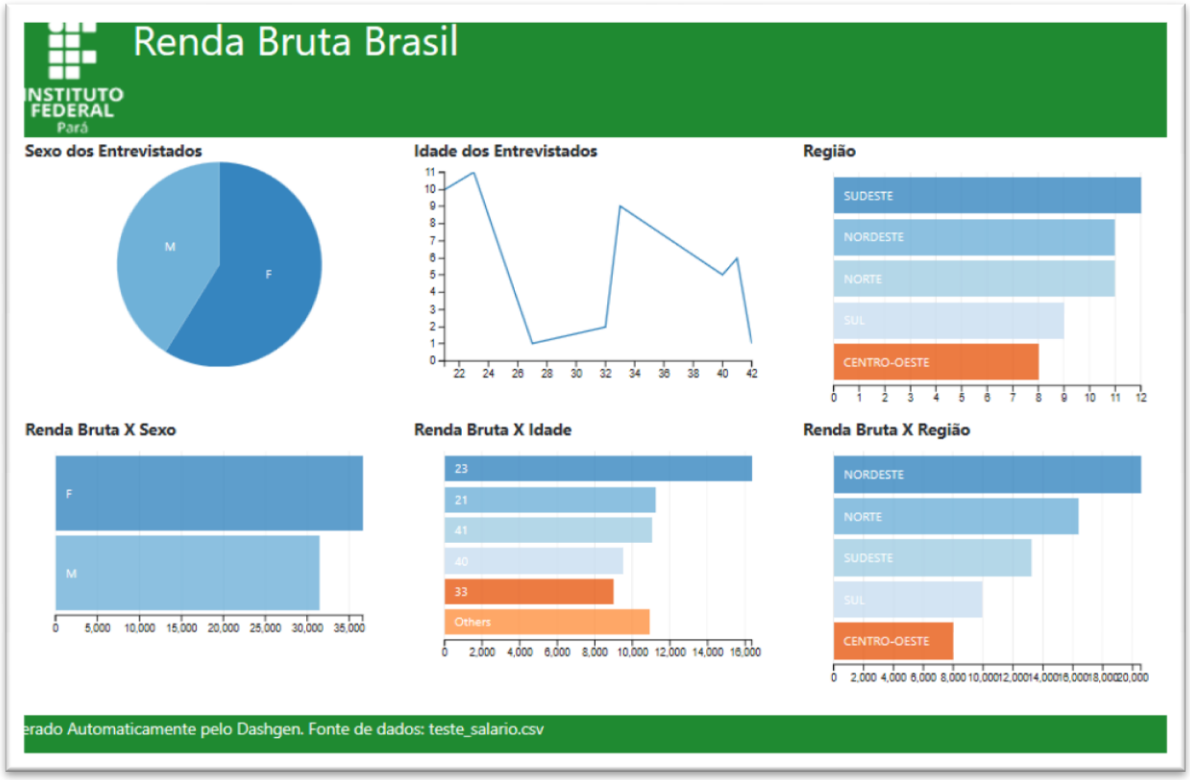
Figura 29 – Confirmação do sucesso na geração do dashboard



Fonte: Elaborada pelo Autor

O dashboard gerado está representado na Figura 30, contendo os seis gráficos nas dimensões e tipos especificados ao DashGen. A composição atendeu perfeitamente às especificações feitas através da interface gráfica de usuário. A aplicação concluiu corretamente a cópia das dependências e a geração do arquivo compactado no diretório de destino. A biblioteca DC.js torna estes gráficos dinâmicos e faz filtros em função das seleções feitas pelo usuário ao clicar sobre um atributo de qualquer dos gráficos.

Figura 30 – Reprodução de tela do Dashboard gerado pelo DashGen



Fonte: Elaborada pelo Autor

4.9 Particularidades do ambiente para execução do DashGen

O projeto DashGen foi concebido para ser distribuído em um pacote JAR. O JAR (Acrônimo para Java ARchive ou arquivo Java) é um formato de arquivo compactado tipicamente usado para agregar classes Java compiladas, metadados e recursos como arquivos contendo texto, imagens ou vídeos em um único arquivo de distribuição (WIKIPEDIA.ORG, 2019e). Os binários são empacotados no JAR, enquanto os arquivos de gabarito e os arquivos de dependência do Dashboard devem ser alocados no mesmo diretório do arquivo JAR, nos subdiretórios *templates* e *boilerplate* respectivamente.

Apesar do fato de o arquivo *frontend.fxml*, que define a interface gráfica JavaFX 8 não ser compilado com o pacote, este foi também adicionado ao JAR, por questões de facilidade de referencia em tempo de execução. A aplicação foi validada nos sistemas operacionais Windows 10 e Xubuntu 18.04 LTS, sem qualquer problema quanto à execução, ambos usando o JRE da Oracle (versão 8u241).

5 CONSIDERAÇÕES FINAIS

A aplicação projetada e desenvolvida como objeto do presente trabalho comprova que a programação generativa baseada em gabaritos é uma alternativa viável na redução do esforço para geração de código e documentos que seguem um padrão preestabelecido. Além disso, fornece uma visão clara de como desenvolver a geração de documentos compostos usando o motor de gabaritos do Apache Freemarker. A classe Java *Gerador*, responsável pela geração do artefato final, assim como a classe *Dataset*, que permite a manipulação do arquivo CSV podem ser alteradas e adaptadas, sem muitas dificuldades, a domínios e contextos diferentes.

O uso da DC.js e suas dependências também se mostrou como uma ferramenta poderosa para a criação de painéis e gráficos com filtros dinâmicos. A velocidade do redesenho dos gráficos a cada definição de filtro é notadamente alta, mesmo usando arquivos com milhares de registros.

Ao longo da pesquisa e desenvolvimento deste trabalho, foi vivenciada a dificuldade que algumas equipes de desenvolvimento encontram ao decidir optar pelo reuso de bibliotecas. Para o emprego das bibliotecas Apache CommonsCSV e Apache FreeMarker, foi necessário estudar a documentação disponível e fazer alguns testes até que se chegasse ao artefato final desejado. Apesar do tempo consumido nessas tarefas ter sido razoavelmente longo, certamente foi menor do que seria necessário para desenvolver essas soluções a partir do início. A mesma observação se aplica à DC.js e suas dependências.

Lidar com fontes de dados como os arquivos em formato CSV, que não possuem um modelo de especificação de metadados, também é uma dificuldade a mais. O uso de uma biblioteca, como a Apache Commons CSV, facilita o tratamento de arquivos CSV mas não possui recursos para a identificação de atributos, problema que foi mitigado com o desenvolvimento de funcionalidades próprias, neste caso os métodos isNumeric() e columnDiscoverType(), que diferenciam atributos com conteúdo de tipo numérico.

O projeto encontra-se disponível para download no repositório github.com, no endereço https://github.com/glaubergad/tccrepo. Lá estão depositados todos os artefatos usados nas etapas de análise e projeto da aplicação, o documento

contendo este trabalho de conclusão de curso e o código fonte do protótipo final DashGen.

Para trabalhos futuros, cabem como sugestões:

a) Refatoração do código fonte, para otimização das relações entre as classes, melhorando o desempenho da aplicação;
b) Transformação do mecanismo de composição em uma biblioteca reutilizável em outros contextos além da geração de *dashboards*;
c) Usando o DashGen como base, criar um artefato nos padrões do projeto Maven, a fim de disponibilizá-lo para a comunidade usuária de software livre, da mesma forma que as bibliotecas utilizadas neste projeto;
d) Aprimoramento do projeto, embarcando inteligência artificial capaz de analisar os dados constantes no arquivo CSV a fim de propor quais seriam as melhores representações gráficas para cada tipo de dado identificado;
e) Permitir maior personalização do Dashboard, como seleção de cores e modelos de tela pré-definidos;
f) Otimização da interface gráfica do usuário, trazendo maior interatividade e tornando-a mais intuitiva.

REFERÊNCIAS

ACADEMIAIN. **QLIK SENSE: O QUE É, COMO FUNCIONA E QUAIS AS VANTAGENS?** Disponível em: <https://blog.academiain1.com.br/qlik-sense-o-que-e-como-funciona-e-quais-as-vantagens/>. Acesso em: 10 jan. 2020.

APACHE.ORG. **The Apache Velocity Project**. Disponível em: <https://velocity.apache.org/>. Acesso em: 7 jan. 2019a.

APACHE.ORG. **Changes Report - Apache Velocity**. Disponível em: <https://velocity.apache.org/engine/devel/changes.html>. Acesso em: 22 ago. 2019b.

APACHE.ORG. **Version History - Apache Freemarker**. Disponível em: <https://freemarker.apache.org/docs/app_versions.html>. Acesso em: 25 ago. 2019c.

APACHE.ORG. **What is Maven**. Disponível em: <https://maven.apache.org/what-is-maven.html>. Acesso em: 1 ago. 2019d.

APACHE.ORG. **Apache Commons CSV**. Disponível em: <https://commons.apache.org/proper/commons-csv/index.html>. Acesso em: 2 nov. 2019e.

APACHE.ORG. **Apache Commons IO**. Disponível em: <https://commons.apache.org/proper/commons-io/>. Acesso em: 10 nov. 2019f.

APACHE.ORG. **Apache Netbeans**. Disponível em: <https://netbeans.org/>. Acesso em: 12 jan. 2019g.

APACHE.ORG. **FreeMarker Java Template Engine**. Disponível em: <https://freemarker.apache.org/index.html>. Acesso em: 1 jun. 2019.

ASTAH. **Astah UML**. Disponível em: <http://astah.net/editions/uml-new>. Acesso em: 10 jan. 2020.

BAELDUNG.COM. **Introduction to Apache Velocity**. Disponível em: <https://www.baeldung.com/apache-velocity>. Acesso em: 7 jan. 2019.

BERGEN, J. VAN. **Velocity or FreeMarker? Two open source Java-based template engines compared**. Disponível em: <https://www.javaworld.com/article/2077797/open-source-tools/velocity-or-freemarker.html>. Acesso em: 29 ago. 2018.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2015.

BOSTOCK, M. et al. **D3.js - Data Driven Documents**. Disponível em: <https://d3js.org/>. Acesso em: 10 nov. 2018.

COSTA, H. M. K. et al. Grandes Massas de Dados na Nuvem: Desafios e Tecnicas para Inovação. **Sbrc 2012**, n. Ouro Preto, MG, Brasil, 2012.

CROSSFILTER ORGANIZATION. **Crossfilter js library**. Disponível em: <https://github.com/crossfilter/crossfilter>. Acesso em: 2 nov. 2019.

CRUZ, S. A. B.; MOURA, M. F. **Formatação de Dados Usando a Ferramenta Velocity**Campinas, 2002.

JETBRAINS. **IntelliJ IDEA**. Disponível em: <https://www.jetbrains.com/idea/>. Acesso em: 12 jan. 2020.

KRUEGER, C. W. Software reuse. **ACM Computing Surveys**, v. 24, n. 2, p. 131–183, 1992.

LUCRÉDIO, D. Uma Abordagem Orientada a Modelos para Reutilização de Software. p. 277, 2009.

LUIZ, A. **Visualização dos dados estatísticos da UERJ : proposta de dashboards baseados no trabalho de Jacques Bertin**. [s.l.] UERJ, 2013.

MAVENREPOSITORY. **Mavenrepository Apache Velocity**. Disponível em: <https://mvnrepository.com/artifact/org.apache.velocity/velocity>. Acesso em: 10 fev. 2019a.

MAVENREPOSITORY. **Mavenrepository Apache Freemarker**. Disponível em: <https://mvnrepository.com/artifact/org.freemarker/freemarker>. Acesso em: 10 jan. 2019b.

MICROSOFT. **O que é o PowerBI desktop**. Disponível em: <https://docs.microsoft.com/pt-br/power-bi/desktop-what-is-desktop>. Acesso em: 12 out. 2019.

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

ORACLE. **Oracle - JavaFX Overview(Release 8)**. Disponível em: <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm>. Acesso em: 2 dez. 2019.

POTENCIER, F. **The flexible, fast, and secure template engine for PHP**. Disponível em: <https://twig.symfony.com/>. Acesso em: 10 jan. 2020.

QLIK. **Qlik Sense - Plataforma de análise de dados**. Disponível em: <https://www.qlik.com/pt-br/products/qlik-sense>. Acesso em: 10 jan. 2020.

RAUDJÄRV, R. **ZT-ZIP**. Disponível em: <https://github.com/zeroturnaround/zt-zip>. Acesso em: 10 nov. 2019.

SHIMABUKURO JUNIOR, E. K. Um Gerador de aplicações configurável. 2006.

SOMATIVA. **Tableau Desktop**. Disponível em: <http://www.somativa.com.br/tableau-desktop>. Acesso em: 12 out. 2019.

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

SYRIANI, E.; LUHUNU, L.; SAHRAOUI, H. Systematic mapping study of template-based code generation. **Computer Languages, Systems and Structures**, v. 52, p. 43–62, 2018.

TABLEAU. **Tableau Desktop**. Disponível em: <https://www.tableau.com/pt-br/products/desktop>. Acesso em: 12 dez. 2019.

TEAM DC.JS. **dc.js - Dimensional Charting Javascript Library**. Disponível em: <https://dc-js.github.io/dc.js/>. Acesso em: 10 nov. 2019.

TUTORIALSPOINT. **Tutorialspoint - DC.js tutorials**. Disponível em: <https://www.tutorialspoint.com/dcjs/>. Acesso em: 11 nov. 2018.

WIKIPEDIA.ORG. **Apache FreeMarker**. Disponível em: <https://en.wikipedia.org/wiki/Apache_FreeMarker>. Acesso em: 25 nov. 2018.

WIKIPEDIA.ORG. **Apache Velocity**. Disponível em: <https://en.wikipedia.org/wiki/Apache_Velocity>. Acesso em: 10 fev. 2019a.

WIKIPEDIA.ORG. **Apache Maven**. Disponível em: <https://pt.wikipedia.org/wiki/Apache_Maven>. Acesso em: 22 ago. 2019b.

WIKIPEDIA.ORG. **JavaFX**. Disponível em: <https://en.wikipedia.org/wiki/JavaFX>. Acesso em: 11 nov. 2019c.

WIKIPEDIA.ORG. **MVC**. Disponível em: <https://pt.wikipedia.org/wiki/MVC>. Acesso em: 11 dez. 2019d.

WIKIPEDIA.ORG. **JAR (file format)**. Disponível em: <https://en.wikipedia.org/wiki/JAR_(file_format)>. Acesso em: 10 jan. 2020e.

WIKIPEDIA.ORG. **Template Processor**. Disponível em: <https://en.wikipedia.org/wiki/Template_processor>. Acesso em: 6 fev. 2020.

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