



UFOP

Universidade Federal
de Ouro Preto

**Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas**

***Framework* para Desenvolvimento de
Processos Baseados na Plataforma
FLUIG**

Mario Soares de Moraes

**TRABALHO DE
CONCLUSÃO DE CURSO**

ORIENTAÇÃO:
Darlan Nunes de Brito

**Julho, 2019
João Monlevade–MG**

Mario Soares de Moraes

***Framework* para Desenvolvimento de Processos
Baseados na Plataforma FLUIG**

Orientador: Darlan Nunes de Brito

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina “Trabalho de Conclusão de Curso II”.

Universidade Federal de Ouro Preto

João Monlevade

Julho de 2019

M827f

Morais, Mario.

Framework para desenvolvimento de processos baseados na plataforma FLUIG [manuscrito] / Mario Moraes. - 2019.

48f.: il.: color.

Orientador: Prof. Dr. Darlan Brito.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Framework (Arquivo de computador). 2. Automação. 3. JavaScript (Linguagem de programação de computador). 4. Padrões de software. I. Brito, Darlan. II. Universidade Federal de Ouro Preto. III. Título.

CDU: 004.41

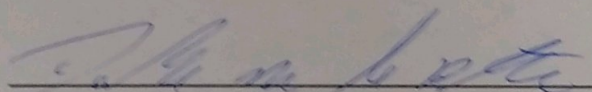
Catálogo: ficha.sisbin@ufop.edu.br

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

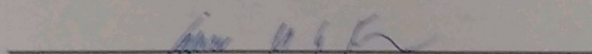
Framework para Desenvolvimento de Processos Baseados na Plataforma FLUIG

Mario Soares de Moraes

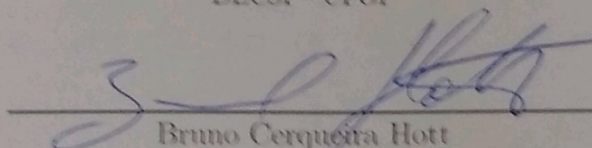
Monografia apresentada ao Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CSI496 – Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação e aprovada pela Banca Examinadora abaixo assinada:



Darlan Nunes de Brito
Prof. Dr.
DECSI - UFOP



George Henrique Godim da Fonseca
Prof. Dr.
Examinador
DECSI - UFOP



Bruno Cerqueira Hott
Prof. Me.
Examinador
DECSI - UFOP

João Monlevade, 12 de julho de 2019

Este trabalho é dedicado à minha mãe Fátima e meu pai Maurício, que me apoiaram durante toda a graduação. Também dedico a minha namorada Luciana, que mesmo de longe, se fez presente, me ajudando chegar ao objetivo.

Agradecimentos

Agradeço a Deus por me dar forças para sempre continuar.

Aos meus pais, que me ajudaram em todas as dificuldades encontradas, não me deixando desanimar.

A minha namorada, que mesmo de longe, deu todo o apoio necessário.

Aos meus irmãos de República, que se tornaram minha segunda família, desde o momento que entrei na faculdade.

Aos amigos de curso, pela amizade e companheirismo.

Aos novos amigos de empresa, que me deram total apoio no final da caminhada.

A universidade Federal de Ouro Preto, pela oportunidade oferecida.

E ao meu orientador, pela orientação, suporte e paciência durante a realização deste trabalho.

Resumo

Para qualquer empresa atingir seus objetivos existem processos que devem ser realizados. Estes, quando executados de forma eficiente, permitem que a empresa produza os resultados almejados, como aumento dos lucros e produtos mais eficientes. Para que os resultados desejados sejam atingidos, estes processos estão sendo automatizados com a intenção de gerenciá-los de forma mais eficiente, ou seja, com o menor trabalho no menor tempo possível. Durante a fase de desenvolvimento de projetos onde são desenvolvidos os programas responsáveis por essa automação, é importante que sejam seguidos padrões e que o desenvolvimento ocorra de forma ágil. Com isto será criado um projeto, que de fato, será útil para as empresas. Além disto, poderá contribuir com outros desenvolvedores de projetos equivalentes na mesma área. Vislumbrando criar uma ferramenta que auxilie os desenvolvedores destes projetos, este trabalho teve como objetivo a construção de um *framework*. Este é capaz de auxiliar no desenvolvimento de uma automação de um processo qualquer, em uma plataforma *online* chamada Fluig. Este *framework* foi criado utilizando a linguagem de programação JavaScript, o arquivo fonte deve ser colocado na pasta do projeto e posteriormente será chamado pelo *script* principal que deverá seguir as regras descritas no decorrer deste trabalho. Com a utilização do *framework*, o desenvolvimento se torna mais simples e rápido, além disso, será possível gerar um padrão de desenvolvimento facilitando a manutenção do código. Outra característica do *framework*, é a possibilidade da criação de novas funcionalidades para o projeto, além das já existentes, evitando a perda de autonomia do desenvolvedor.

Palavras-chaves: *Framework*. JavaScript. Processo.

Abstract

Companies need to complete their processes in order to achieve their goals. These processes allow the company to gather the desired results when performed efficiently, such as increase in profits and products more efficient. In order to accomplish it, these processes are being automated in so it is possible to achieve better quality. The automation of these processes needs to follow certain design patterns while also be agile. This research created a framework in JavaScript language, capable in assist the development of the automatization of any process. In order to use it, the developer needs to place a file in the project folder, which will be called by the main script, as described by the rules described in the course of this work. With the use of this tool, the development becomes simpler and faster, in addition, it will be possible to generate a development pattern that helps code maintenance. It is also possible to add new functionalities to this project in addition to existing ones, avoiding the loss of developer autonomy.

Key-words: framework. javascript. process.

Lista de ilustrações

Figura 1 – Diagrama de gerenciamento em um <i>software</i> ERP.	13
Figura 2 – Processo de admissão	14
Figura 3 – Página <i>home</i> da plataforma Fluig	18
Figura 4 – <i>Cards</i> da plataforma Fluig	19
Figura 5 – Fluxo BPMN	20
Figura 6 – HTML - CSS - JavaScript	21
Figura 7 – Execução do <i>framework</i>	26
Figura 8 – Fluxo projeto exemplo	39
Figura 9 – Formulário na atividade Início	42
Figura 10 – Campo Data Exemplo	43
Figura 11 – Campo Monetário Exemplo	43
Figura 12 – Campo Consulta Exemplo	43
Figura 13 – Regra de obrigatoriedade do campo Consulta Exemplo	44
Figura 14 – Tabela pai e filho	44
Figura 15 – Começo do formulário na atividade Aprovação	45
Figura 16 – Final do formulário na atividade Aprovação	45
Figura 17 – Seção Aprovação	46
Figura 18 – Ação função customizada	46
Figura 19 – Começo do formulário na atividade Fim	46
Figura 20 – Final do formulário na atividade Fim	46

Lista de abreviaturas e siglas

ERP *Enterprise Resources Planning* ou Planejamento dos Recursos da Empresa

BPM *Business Process Management* ou Gerenciamento de Processos de Negócio

BPMN *Business Process Model and Notation*

IDE *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado

HTML *Hypertext Markup Language* ou Linguagem de Marcação de Hipertexto

CSS *Cascading Style Sheets* ou Folha de Estilo em Cascatas

Sumário

1	INTRODUÇÃO	12
1.1	O problema de pesquisa	13
1.2	Motivação	14
1.3	Objetivos	15
1.4	Organização do trabalho	16
2	CONCEITOS GERAIS E REVISÃO BIBLIOGRÁFICA	17
2.1	Fluig	17
2.2	<i>Business Process Management</i> (BPM)	18
2.3	Base do Projeto BPM Fluig	20
2.3.1	HTML	21
2.3.2	CSS	21
2.3.3	JavaScript	22
2.4	Outras Ferramentas	22
2.4.1	Bootstrap	23
2.4.2	Mustache	23
2.4.3	Moment.js	23
2.4.4	jQuery	23
3	DESENVOLVIMENTO	25
3.1	Criação do Framework	25
3.1.1	Objeto Principal	25
3.1.2	Objeto <i>globalFunctions</i>	27
3.1.3	Objeto <i>fields</i>	27
3.1.4	Objeto <i>sections</i>	30
3.1.5	Objeto <i>tables</i>	30
3.1.6	Objeto <i>customActions</i>	31
3.1.7	Objeto <i>defaults</i>	31
3.1.8	Objeto <i>utils</i>	32
3.1.9	Funções Reaproveitadas	33
3.2	Manual de Utilização	33
3.2.1	Array <i>fieldsConfig</i>	34
3.2.2	Array <i>sectionsConfig</i>	36
3.2.3	Array <i>tablesConfig</i>	36
3.2.4	Array <i>customActionsConfig</i>	37

4	RESULTADOS	39
4.1	Projeto Exemplo com <i>Framework</i>	39
4.2	Execução do Projeto Exemplo com <i>Framework</i>	42
4.2.1	Atividade Início	42
4.2.2	Atividade Aprovação	43
4.2.3	Atividade Fim	44
5	CONCLUSÃO	47
	REFERÊNCIAS	48

1 Introdução

Um *software* de gestão de informações é importante para empresas de todos os tamanhos e ramos de atividade. Este tipo de *software* é responsável por gerenciar informações relativas ao controle financeiro, estoque, funcionários, dentre outros. Devido à popularidade deste tipo de programa surgiram um grande número de empresas direcionadas à confecção deste tipo de *software*. Estes sistemas de gestão de informações são comumente chamados de *Enterprise Resources Planning* ou Planejamento dos Recursos da Empresa (ERP). O ERP, segundo (HABERKORN, 2003): “Chegou ao Brasil na década de 90, quando aqui aportaram as empresas estrangeiras do setor.” Também segundo (HABERKORN, 2003):

ERP é um sistema de gestão para empresas que propõe um tipo de solução: a informatização integrada de todos os processos de uma empresa, sejam eles contábeis, financeiros, de recursos humanos, de estoques, custos, compras, produção, faturamento e etc.

Com isto, os ERPs se tornaram fundamentais para todos os tipos de empresas, inclusive as de pequeno porte. Para que seja possível que estas informações estejam disponíveis em qualquer lugar são necessárias varias coisas e o mais importante delas é um meio de transmissão de dados binários e este meio é a rede mundial de computadores, chamada de internet.

Os ERPs são importantes ferramentas utilizadas na administração das empresas. A [Figura 1](#) apresenta uma representação esquemática do que é um ERP. Sendo ferramentas de administração, é uma condição desejável que seja possível acessar as informações contidas nos ERPs a todo momento e em qualquer lugar, mas sem comprometer a segurança dos dados armazenados. Para atingir este objetivo, foram criadas diversas plataformas *online* integradas com o sistema de gestão. Além do acesso *online* e outras vantagens, estas plataformas trazem, uma interface mais agradável e de fácil utilização para o usuário final. Os softwares ERPs, são normalmente difíceis de serem utilizados, principalmente quando o usuário tem pouco conhecimento técnico ou teórico em computação, estas plataformas atuam no sentido de diminuir esta distância entre o usuário final e as informações necessárias. Um fator importante é que mesmo utilizando estas plataformas para programação e interface com o usuário a segurança dos dados continua mantida. Os ERPs exigem autenticação para liberar acesso e como o usuário tem acesso apenas à plataforma, os dados ficam protegidos em mais uma camada de software, fazendo com que aumente a segurança.

Um grande exemplo destas plataformas utilizadas para confecção deste tipo de software, interligados com os ERPs, é a plataforma FLUIG, desenvolvida pela TOTVS, que possui todos os direitos autorais. Sendo o fluig uma ferramenta onde diversas ferramentas



Figura 1 – Diagrama de gerenciamento em um *software* ERP.

Fonte: site <http://projetosae5.blogspot.com/p/erp-sistema-de-planejamento-de-recursos.html>.

são implementadas em um mesmo local para diversos usuários.

Esta plataforma possui uma série de ferramentas que auxiliam no trabalho diário da empresa, como por exemplo: Gestão de documentos, criação de páginas públicas ou privadas, social interno entre colaboradores, gestão de processos, entre outros. Esta plataforma permite utilizar programação em linguagem JavaScript, para customizar qualquer uma das ferramentas. Esta grande versatilidade na utilização desta plataforma nos inspirou a criar um novo conjunto de bibliotecas, para auxiliar no desenvolvimento na área de gestão de processos na plataforma Fluig.

1.1 O problema de pesquisa

Sendo assim o que procuramos desenvolver neste trabalho, é uma ferramenta que vai auxiliar os desenvolvedores na plataforma Fluig. A esta ferramenta chamamos de *framework*. Para utilização deste *framework* a plataforma Fluig utilizada deverá estar na versão 1.5.0 ou superior. Com o *framework* desenvolvido neste trabalho, vai ser possível utilizar um *script* previamente construído do *framework*, para criar todas as funcionalidades básicas do projeto. Para isto é suficiente passar as configurações desejadas, deixando para o desenvolvedor, implementar as funcionalidades que são mais específicas de cada projeto. As funcionalidades básicas do projeto podem ser processos, como por exemplo, uma admissão

de funcionário, onde é necessário um cadastro de dados, uma aprovação e lançamento de dados no ERP entre outros. A [Figura 2](#), apresenta um exemplo de processo que poderá ser criado na plataforma, que deve ser na linguagem JavaScript, versão ECMAScript 2015 ou superior. O desenvolvedor precisa criar todas as funcionalidades, sendo estas, as regras de cada etapa do processo.

Figura 2 – Processo de admissão



Fonte: Produzido pelo autor.

1.2 Motivação

Uma das utilidades de integrar a plataforma online com o ERP, é automatizar um processo, que em alguns casos, já é realizado na empresa de forma manual. Onde a parte de cadastro é feito por preenchimento em papel e então estes dados são lançados em algum *software* de gestão. Quando realizado na plataforma este processo deve se tornar mais rápido e simples, eliminando a necessidade de utilização de papel já que os dados podem ser lançados diretamente na plataforma para que estes dados seja posteriormente aprovados e enviados para o ERP. Estas etapas de cadastro dos dados, aprovação e lançamento destes no sistema, fazem parte de um processo, que é algo mais amplo que envolve os fins pelos quais os dados são lançados no sistema, por exemplo, a contratação de um funcionário.

O conceito de processo tem sido adotado por várias empresas e disseminado pelos gestores e equipes. Assim, a terminologia Processo já é conhecida dos gestores. Está relacionado ao conceito de uma série de atividades que, em geral, possui uma sequência relativamente lógica e que tem um objetivo claro a ser alcançado, que pode ser um produto ou serviço a ser produzido e entregue ([SILVA, 2017](#)).

Para ajudar no planejamento e execução dos processos, existe uma ferramenta administrativa chamada de *Business Process Management* ou Gerenciamento de Processos de Negócio (BPM). No caso deste trabalho, BPM é de fundamental importância, pois a plataforma Fluig, utiliza seus conceitos, ao automatizar um processo. Na prática, sem pensar na utilização na plataforma Fluig, o BPM que fornece um conjunto de conceitos, que quando bem utilizados, ajudam a criar as estratégias para se criar e executar um processo.

Dentre os vários conceitos que o BPM fornece, o mais importante para a elaboração deste trabalho é o *Business Process Model and Notation* (BPMN), de modo objetivo segundo (SILVA, 2017), “BPMN é um conjunto de elementos e regras que compõem um modelo e é, ao mesmo tempo, uma notação para representar processos de negócio.” Esta notação gráfica, mostrada na Figura 2, representa um fluxo na plataforma Fluig, onde em cada etapa desse fluxo, é executado um trecho de código criado pelo desenvolvedor. A criação deste fluxo poderá ser modificada para melhor se adequar às necessidades do usuário, aplicando suas regras de negócio de uma maneira simples para quem está utilizando o sistema.

Vendo os conceitos de ERP, plataformas online e BPM, é possível perceber que os mesmos estão interligados, pois, existindo um processo em uma empresa, é possível criar seu diagrama BPMN. Este diagrama é implementado em uma plataforma, onde o usuário final ou solicitante tem acesso e que seja integrado com o ERP, onde de fato ficam guardados os dados da empresa, mas o solicitante não tem acesso. O problema de pesquisa abordado neste trabalho é o desenvolvimento de um *framework* que facilite o desenvolvimento de programa para automação de processos na plataforma Fluig e construa um padrão de programação, diminuindo complexidade de desenvolver nesta.

1.3 Objetivos

O presente trabalho consiste em desenvolver um *framework* que auxilia no desenvolvimento de um processo com fluxo BPMN, na plataforma Fluig, que pode ou não ser integrado com o software chamado RM, sendo este, o ERP desenvolvido pela TOTVS. Ao desenvolver na plataforma Fluig, existem regras que devem ser seguidas. No entanto, estas regras não são rígidas permitindo ao desenvolvedor utilizar inúmeras maneiras diferentes de implementar o software de gerenciamento de um processo. Estas diferentes maneiras podem funcionar corretamente, mas podem não ser a maneira correta de desenvolver, para se ter uma boa usabilidade e performance. A motivação de construir o *framework*, é apoiar o desenvolvedor para que ele possa seguir padrões de desenvolvimento, diminuindo assim sua dificuldade, fazendo com que a solução proposta no projeto, seja adequada para a plataforma Fluig. Com a utilização do *framework*, o desenvolvimento de um programa para automatização de um processo, como admissão de funcionário, solicitação de férias, aprovação de compra, entre outros, seja mais fácil e rápido. Além disso, como vai gerar uma padronização, a manutenção do código fonte, mesmo depois de um certo tempo, será de rápida compreensão para qualquer membro da equipe, mesmo que este não tenha participado ativamente do projeto. Para a empresa que adotar o uso do *framework*, será mais fácil dar treinamento para novos membros da equipe. Há ainda outra grande vantagem que é ter um produto final com mais qualidade e agilidade. Além de levar a um tempo menor de desenvolvimento do que seria necessário para implementar caso o trabalho fosse

realizado sem o auxílio do *framework*.

Este trabalho possui aos seguintes objetivos específicos:

- Desenvolver o *framework*, utilizando a linguagem JavaScript.
- Desenvolver um projeto, com um arquivo HTML e outro JavaScript, que utilize o *framework* e sirva de base para os desenvolvedores.

1.4 Organização do trabalho

O restante deste trabalho é organizado como se segue. O Capítulo 2, apresenta os conceitos gerais e revisão bibliográfica para ser possível entender melhor os principais tópicos abordados neste trabalho. O Capítulo 3 foi dividido em duas seções. A Seção 3.1, explica como foi desenvolvido o *framework*, mostrando a funcionalidade de cada objeto presente no *script*. Já a Seção 3.2 mostra como se deve utilizar o *framework*. O Capítulo 4, também foi dividido em duas seções, a Seção 4.1, mostra a criação de um projeto utilizando o *framework* e a Seção 4.2, mostra o projeto sendo executado na prática. Por fim, o Capítulo 5, explica o que foi alcançado e mostra o que pode ser feito em trabalhos futuros.

2 Conceitos Gerais e Revisão Bibliográfica

O conceito de *framework* está presente em diversas áreas que envolvem programação. Existe um grande número destas ferramentas que são utilizadas especificamente para programação de aplicações web, podemos citar como exemplo destes: React, Vue.js, AngularJS, Node.js, jQuery, entre outras, utilizadas apenas com o objetivo de facilitar no desenvolvimento quando é utilizada a linguagem JavaScript. Uma destas ferramentas que tem grande importância neste projeto é o jQuery, que será utilizada durante o desenvolvimento do *framework* proposto neste trabalho. Durante a revisão da literatura, não foi encontrada nenhuma outra ferramenta, que tenha objetivo central igual ao que estamos propondo, sendo este, de auxiliar no desenvolvimento de um processo, utilizando fluxo BPMN na plataforma Fluig. O principal motivo de ainda não existir uma ferramenta com este objetivo, se deve ao fato de o Fluig ser uma plataforma nova, com apenas cinco anos de criação. Mas reforçamos que apesar de nova já é bastante utilizada com uma quantidade estimada de duas mil empresas em 2017 que a utilizavam. Com isto, neste capítulo vão ser apresentados os principais conceitos sobre onde e como o *framework* vai ser desenvolvido.

2.1 Fluig

Fluig é uma plataforma, *online*, para criação de programas na WEB que têm como objetivo automatizar as tarefas de uma determinada empresa. Esta plataforma, é então utilizada na criação de programas para uma empresa, esta adquire um número limitado de licenças para usuários e fazem as customizações necessárias para atender suas necessidades específicas. A partir deste ponto, os programadores podem utilizar a plataforma para criar os programas e os funcionários da empresa para visualizar e lançar dados no sistema. Tantos programadores e funcionários utilizam a plataforma, tendo acesso a estes conteúdos, o que permite personalizar quase tudo que necessitam para seu trabalho diário.

Quando o usuário acessa a plataforma, a primeira página mostrada é a *home* do Fluig, que é mostrada na [Figura 3](#). Esta página principal pode ser customizada e a partir dela, é possível ter acesso a todas as outras funcionalidades presentes no Fluig. Segundo entrevista do Sr. Marcio Martucheli, Diretor de Atendimento e Relacionamento Serviços TOTVS Vale do Paraíba para o artigo ([CAMARGO et al.,](#)): “Fluig é uma plataforma robusta que une todas as ferramentas de trabalho de uma empresa onde se acessa processos (BPM) a partir de um único login, totalmente integrados aos sistemas de gestão da sua empresa.” Quando a organização adquire o Fluig, recebe todas as funcionalidades presentes na plataforma, que os criadores chamam de *cards*. A [Figura 4](#) ilustra esses *cards*. Não é o

foco do trabalho explicar cada uma dessas funcionalidades, na Seção 2.2 vai ser apresentado mais detalhadamente o *card* BPM - Gestão de Processos, que é a área de atuação do *framework*.

Figura 3 – Página *home* da plataforma Fluig



Fonte: <<http://fluig.com/>>

2.2 Business Process Management (BPM)

Em toda empresa ou organização, existem inúmeros trabalhos diários para serem realizados, desde o mais simples até o mais complexo. Para o bom desempenho destes trabalhos, é necessário planejar as atividades, ter uma boa organização e foco no que precisa ser feito. Quando isto não é alcançado todo o trabalho pode ser perdido.

Tendo em vista o fato das empresas não terem a disciplina de definir seus processos, conforme (SILVA, 2017), o BPM tem a intenção ajudar a modelar os mesmos, abordando o trabalho a ser realizado do começo ao fim, fazendo a análise, modelagem e desenho de todas as etapas a serem realizadas. Dentre todos os conceitos, o fluxo BPMN é a forma visual de se representar o processo, (MOCROSKY, 2012) define BPMN como, “uma notação gráfica para captura do processo de negócio, usada para documentar e promover a comunicação entre os processos dentro e fora da empresa e partes interessadas.”

Para a construção de um fluxo BPMN, não são necessários muitos elementos, basicamente no diagrama, existem as atividades, *gateways* e fluxo de sequência. A Figura 5 ilustra um exemplo.

Figura 4 – Cards da plataforma Fluig

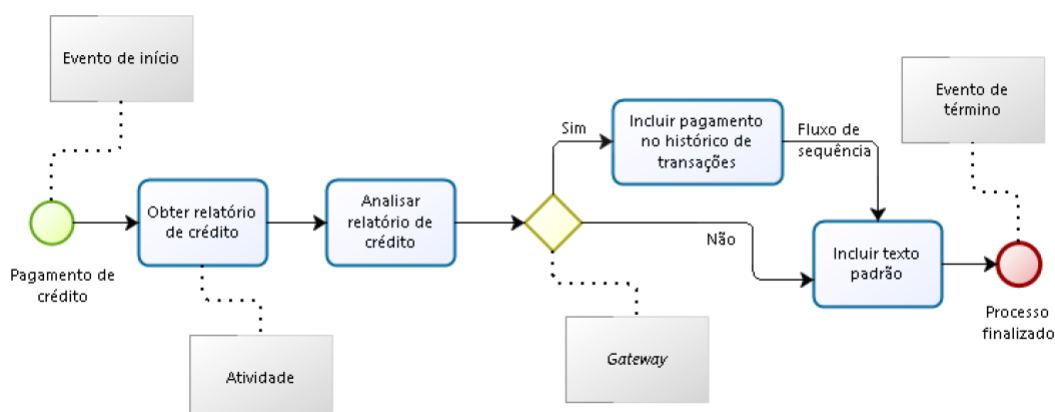


Fonte: <<http://tdn.totvs.com/display/public/fluig/Licenciamento+fluig/>>

- Atividades: Tarefas ou ações do processo, incluindo o início e fim. São representadas pelos retângulos, atividade inicial por círculo verde e atividade final por círculo vermelho.
- *Gateways*: Utilizados quando é necessário algum tipo de decisão que deve dividir o fluxo, são representados pelos losangos.
- Fluxo de Sequência: São as ligações, entre o início e fim do processo, passando pelas atividades e *gateways*, são representados pelas setas.

Depois da construção do fluxo BPMN do processo, será necessário fazer a automação. Para se fazer esta automação, é utilizado o *card* BPM - Gestão de Processos do Fluig. Para fazer esta construção deve ser utilizada uma *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado (IDE), que tenha comunicação com a plataforma. É criado um projeto nesta IDE e todo o desenho do fluxo que acabou de ser mencionado, é programado baseado no projeto. Associado a este fluxo, deve ser desenvolvido um formulário e um *script* em linguagem JavaScript. A ideia final é que cada etapa do fluxo desenvolvido, seja desempenhada por um responsável ou grupo de responsáveis pré definidos. Nestas atividades, vão existir ações aplicadas ao formulário. Durante cada etapa, deve ser exibido a parte do formulário que é de interesse do responsável daquela atividade, podendo ser ou não editável por esta pessoa. Com o processo criado na plataforma, o usuário consegue

Figura 5 – Fluxo BPMN



Fonte: (SILVA, 2017, p. 252)

ter controle de todas as solicitações que estão em aberto para ele, dependendo dele dar continuidade ao fluxo e pode existir um usuário gestor, para acompanhar todos os processos abertos e finalizados da empresa.

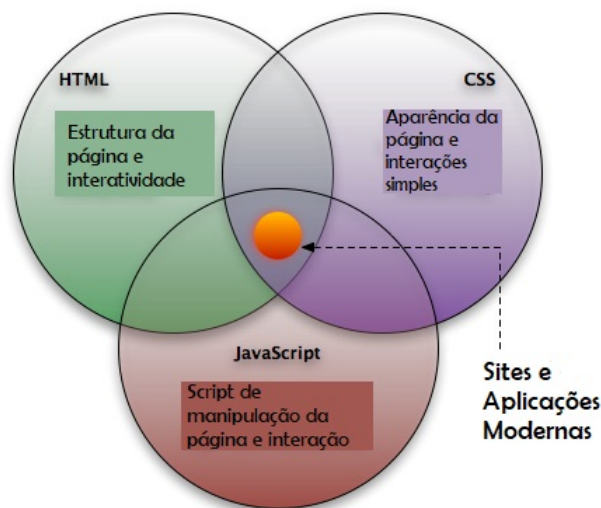
Para ficar um pouco mais claro do motivo de criar um processo BPM no Fluig, podemos considerar o exemplo de admissão de um colaborador. Na atividade inicial do fluxo, o responsável pela admissão, vai poder preencher todos os dados do colaborador a ser admitido, depois disso, o processo pode ser enviado para outra pessoa, que vai poder visualizar os dados e tomar a decisão de admitir o colaborador, negar a admissão ou devolver para o usuário inicial, informando que algum dado precisa ser corrigido. Pensando nesse pequeno exemplo para uma admissão, pode não fazer nenhuma diferença a utilização da plataforma, pois tudo poderia ser feito via email ou até mesmo, papel impresso, mas levando em conta uma empresa com várias filiais em locais diferentes, o processo automatizado pode gerar vários tipos de melhorias. Estas melhorias diárias nas empresas que está fazendo o Fluig ter uma boa aceitação.

2.3 Base do Projeto BPM Fluig

Como dito na Seção 2.2, para desenvolver um projeto BPM na plataforma e fazer as manipulações com o formulário e dados, é necessário os arquivos básicos. Para a estruturação do formulário, é utilizado um arquivo no formato *Hypertext Markup Language* ou Linguagem de Marcação de Hipertexto (HTML). Quando é necessário a aplicação de um estilo, é utilizado o formato *Cascading Style Sheets* ou Folha de Estilo em Cascatas (CSS) e um *script* em JavaScript para toda a ação dos elementos, como campos e botões. Com estes três arquivos é possível criar inúmeras possibilidades de projetos com uma excelente

qualidade, a [Figura 6](#) ilustra o objetivo de cada tipo de arquivo.

Figura 6 – HTML - CSS - JavaScript



Fonte: <<https://alineinfo.wordpress.com/tag/html-css/>>

2.3.1 HTML

A linguagem HTML é utilizada pra definir a estrutura da página. Através das suas marcações, chamadas de *tags*, que o navegador vai saber o que deve ser exibido para o usuário.

Sendo uma linguagem de marcação, HTML é utilizado principalmente para estruturar páginas *web*, no caso de projetos BPM no fluig, é de fundamental importância, pois dentro desta estruturação que vai estar contido o formulário base do projeto, que deve ser exibido para o usuário e utilizado para guardar as informações fornecidas pelo mesmo ou consultadas de algum outro ambiente. Dentro deste formulário que devem estar as seções, que vão ser exibidas ou ocultadas e os campos, que vão ser editáveis ou bloqueados, dependendo da atividade atual do fluxo BPM.

2.3.2 CSS

Utilizado para separar a estrutura principal do estilo, o CSS é responsável pela aparência da página. No arquivo CSS que são definidas as cores, fontes e layout para determinar a aparência da página.

O CSS é aplicado ao formulário do projeto, customizando o mesmo com um estilo desejado. Classes devem ser colocadas nas *tags* do HTML e o CSS deve ser construído para customizar os elementos com estas classes. Quando é falado em estilo, não são apenas as

cores do formulário, o CSS pode definir, entre outras propriedades, o tamanho, as formas e até animações dos elementos.

2.3.3 JavaScript

A linguagem de programação JavaScript é o ponto principal desta seção, o motivo para isso, é o *framework* ser desenvolvido nesta linguagem. O desenvolvimento na plataforma Fluig é baseado em JavaScript e tanto a programação *frontend* quanto *backend* podem utilizá-la sem encontrar maiores dificuldades, isso ocorre, pois o servidor da plataforma interpreta a linguagem no *backend* e no *frontend*, os interpretadores já estão embutidos no próprio navegador.

JavaScript foi criada pela Netscape em parceria com a Sun Microsystems, com a finalidade de fornecer um meio de adicionar interatividade a uma página *web*. A primeira versão, denominada JavaScript 1.0, foi lançada em 1995 e implementada em março de 1996 no navegador Netscape Navigator 2.0 quando o mercado era dominada pela Netscape. (SILVA, 2010).

Seguindo a ideia do javascript ser o responsável pela manipulação da página, no desenvolvimento do processo BPM da plataforma Fluig, a linguagem é utilizada, para fazer a iteração com o usuário e fazer a manipulação dos dados, como por exemplo:

- Ao clicar em um botão X, aparece um modal na tela com uma mensagem Y.
- Ao clicar em um campo do tipo data, aparece um calendário na tela para usuário selecionar uma data de maneira mais intuitiva.
- Depois do clique em salvar, guardar os dados informados pelo usuário em uma tabela específica.
- Definir a obrigatoriedade de um campo X, dependendo de uma condição Y.

2.4 Outras Ferramentas

Para auxiliar no desenvolvimento *web*, existem diversas ferramentas que podem atuar desde a estruturação HTML, até no *script* JavaScript, a intenção desta seção é apresentar algumas delas, pois, juntando várias em um mesmo projeto, costuma deixar o desenvolvimento menos complexo. Estas ferramentas tem o mesmo objetivo do *framework* deste trabalho, de ajudar em alguma etapa do desenvolvimento, algumas delas, vão ser utilizadas para o auxiliar no desenvolvimento do *framework*.

2.4.1 Bootstrap

Bootstrap também é um *framework* e tem a intenção de já aplicar um estilo no HTML construído, sem ter que ficar criando o CSS a partir do zero. Na prática, é como se os estilos já estivessem embutidos nas *tags* da estrutura com um padrão, utilizando o código fonte, apenas se for necessário uma alteração nos padrões, pois deixar o HTML puro, com *layout* nativo, não costuma ser a melhor opção, já que a estrutura se torna muito "quadrada". Utilizar o Bootstrap, não significa que o estilo está pronto e não será necessário mais nenhum tipo de aplicação CSS, o mesmo te oferece uma base pronta, que pode e deve ser customizada de uma maneira mais simplificada.

2.4.2 Mustache

Mustache é um sistema de *template* web, também é utilizado na exibição final do conteúdo para o usuário, mas de uma maneira bem diferente do Bootstrap. Utilizando esta ferramenta, é possível de maneira fácil, mostrar na tela um conteúdo, que é criado dinamicamente, ou seja, o que vai ser exibido, não é fixo, pode variar e vai ser criado no momento que o conteúdo for ser renderizado.

Pensando no desenvolvimento na plataforma fluig, fica mais fácil de imaginar um exemplo. Como já mencionado, todo o conteúdo HTML do formulário de um processo, fica dentro de uma única *tag* form. Com a utilização do Mustache, é possível criar um conteúdo fora desta *tag* e com o auxílio da linguagem JavaScript, são inserido parâmetros no Mustache, este conteúdo vai ser exibido dentro da *tag* form, com as informações necessárias naquele momento.

2.4.3 Moment.js

Moment.js é uma biblioteca para utilização na linguagem JavaScript, com objetivo de trabalhar com datas, para (OLIVEIRA et al., 2016): “A biblioteca Moment.js, contém uma variedade bastante grande de operações relacionadas a datas”. Existe uma biblioteca nativa da linguagem, mas na maioria dos casos, sua utilização não é trivial. Com o Moment.js é possível fazer a manipulação com datas de uma forma fácil, podendo ser resgatada a data atual com todas as variáveis, escolher o formato que se deseja trabalhar, fazer operações, como por exemplo somar e subtrair dias ou anos, entre outras, resumindo, é uma biblioteca completa que facilita o trabalho do desenvolvedor.

2.4.4 jQuery

De todas as ferramentas apresentadas, jQuery é a mais importante para o desenvolvimento do *framework* e seu uso é quase indispensável para desenvolver na plataforma Fluig, segundo (COGO et al., 2009) que apresentou uma análise das principais APIs

em linguagem Javascript para à criação de interfaces Web, “Destacam-se os exemplos disponibilizados pela jQuery, que na maioria das vezes são simples, objetivos e possuem uma vasta documentação”. jQuery é uma biblioteca com funções da linguagem JavaScript, estas funções são utilizadas para interagir com a estrutura HTML. Apesar de parecer que jQuery ajuda na exibição do conteúdo para o usuário final, já que atua na estrutura, seu foco principal não é esse, mas sim, ajudar no desenvolvimento quando se é necessário atuar em um elemento específico.

Para facilitar o entendimento, é mais interessante citar alguns exemplos. Com jQuery, é possível selecionar um campo com a *tag input* pelo seu nome, id, atributo ou qualquer classe presente nesta *tag*, depois de selecionar, várias operações podem ser realizadas, como resgatar seu valor, atribuir um valor, adicionar uma classe ou atributo, entre outras. Outro exemplo, seria selecionar todas as linhas de uma tabela e realizar alguma operação sobre elas, de maneira individual ou em todas da mesma forma.

3 Desenvolvimento

Nesta seção serão descritas as etapas que foram seguidas até a finalização deste trabalho. Em uma etapa inicial foi feito um planejamento do *framework* para que nas etapas seguintes houvesse o menor número de retrabalho possível. A próxima etapa foi de implementação dos *scripts* para o funcionamento do *framework* e exemplo de utilização. Em uma última etapa foram realizados testes que levaram a conclusão do trabalho. A seguir serão descritas estas etapas como parte do desenvolvimento deste trabalho.

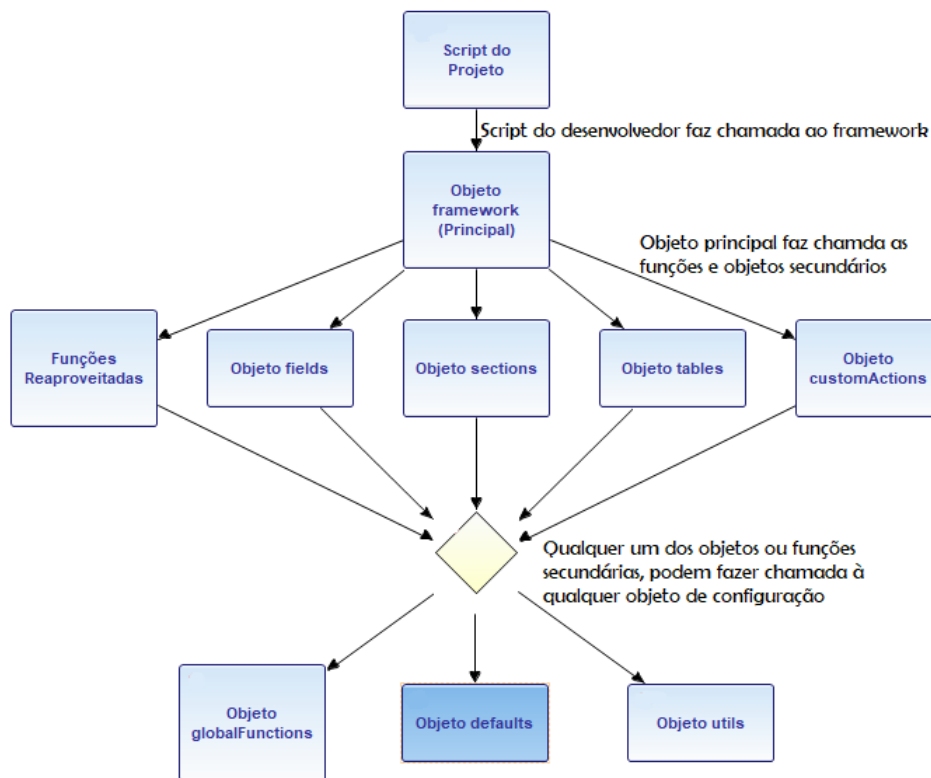
3.1 Criação do Framework

A primeira etapa do projeto, foi realizar o planejamento do *framework*, como seria desenvolvido. Ficou estabelecido que o mesmo seria criado em um único arquivo, utilizando a linguagem JavaScript, para que o *software* tenha bom desempenho e utilize a linguagem reconhecida pela plataforma. Também ficou definido que o *script* seria dividido em objetos, com funções específicas para atuarem em uma área do formulário, para que assim, o código seja de fácil entendimento e não gere nenhum tipo de problema ao realizar uma melhoria ou manutenção. A partir do código fonte do projeto, estes objetos vão chamando uns aos outros para que no final tudo tenha sido executado na ordem correta, como mostra a [Figura 7](#). Nesta seção, vai ser apresentado cada um desses objetos desenvolvidos no *framework*, explicando o modo que foi criado e suas funcionalidades.

3.1.1 Objeto Principal

Foi desenvolvido um objeto chamado *framework*, este nome foi dado, pois ele é o objeto principal, que engloba todos os outros objetos presentes no código fonte, fora desse objeto, estão apenas algumas funções que são utilizadas para algum objetivo específico, sendo elas: *setDisabled*, *fWZoom*, *setDefault*s e uma função que tem o objetivo de fazer uma tradução e por padrão da linguagem, não possui um nome. Estas funções serão detalhadas na Subseção 3.1.9. Dentro deste objeto *framework*, estão todos os outros objetos e a função chamada *init*, que é a função principal do *framework* que deve ser chamada pelo *script* do formulário.

Ao chamar a função *init*, devem ser passados os parâmetros: Estado da Atividade, Número da Atividade, Configuração dos Campos, Configuração das Seções, Configurações das Tabelas e Funções Customizadas. A maneira de como deve ser chamada e o que são estes parâmetros, vão ser mostrados na Seção 3.2. O objetivo desta função é receber todos estes parâmetros e repassa-los para as funções presentes em outros objetos, que serão

Figura 7 – Execução do *framework*

Fonte: Produzido pelo autor.

detalhados nas próximas seções, mas, para deixar claro, é importante já ressaltar, que cada um destes objetos, também possuem uma função chamada *init*, que vão continuar a execução do *framework*. Em ordem de execução, a função *init* principal, chama as seguintes funções:

- *Init* presente no objeto *globalFunctions*, apresentada na Subseção 3.1.2.
- *Init* presente no objeto *fields*, apresentada na Subseção 3.1.3.
- *Init* presente no objeto *sections*, apresentada na Subseção 3.1.4.
- *Init* presente no objeto *tables*, apresentada na Subseção 3.1.5.
- *Init* presente no objeto *customActions*, apresentada na Subseção 3.1.6.
- Por fim, executa uma função que aguarda 100 milissegundos para começar executar e força haver uma mudança no formulário, para que seja feita uma validação em todos os campos.

3.1.2 Objeto *globalFunctions*

O objeto *globalFunctions* foi criado com o objetivo de conter todas as funções padrões que sempre vão ser chamadas quando for utilizado o *framework*. Neste primeiro momento, foi criada apenas a função *init*, mas, em etapas de melhorias do *framework*, será necessária a criação de outras funções consideradas globais.

Init é a função chamada pela função *init* principal do *framework*, onde deve ser passado o parâmetro: Estado da Atividade. Esta função tem o objetivo de criar uma outra função, que vai ser executada toda vez que um campo do formulário for editado e com o auxílio do *plugin* jQuery vai realizar a validação de todos os campos, utilizando também uma função apresentada na Subseção 3.1.9. O interessante desta função ser criada, é que toda vez que o usuário editar um campo, o mesmo vai ficar verde ou vermelho, deixando claro se o campo foi preenchido da forma correta ou não.

3.1.3 Objeto *fields*

O objeto *fields* foi criado para conter todos os objetos e funções que atuam nos campos do formulário. Ao contrário do objeto *globalFunctions* que possui apenas uma função, este objeto contém duas funções (*init* e *start*) e cinco objetos (*zoom*, *money*, *date*, *aprovacao* e *customActions*).

Init é a função chamada pela função *init* principal do *framework*, onde devem ser passados os parâmetros: Estado da Atividade, Número da Atividade e Configuração dos Campos. O objetivo desta função é realizar um *loop* em todas as configurações de campos recebidas, verificar se configuração deve ser aplicada no estado e número da atividade atual do formulário e passando por todas estas condições, chamar a função *start*.

Start é a função chamada pela função *init* do Objeto *fields*, onde devem ser passados os parâmetros: Configuração do Campo e Sufixo do Nome do Campo (se necessário). O objetivo desta função é chamar a função que cria regra de validação do campo independente do tipo de campo, verificar o tipo do campo e chamar a função *init* deste tipo de campo, chamar a função que adiciona classes no campo e chamar função que aplica função customizada no campo.

Zoom é o objeto que contém as funções que atuam nos campos do formulário do tipo consulta. Neste tipo de campo, vai existir um botão ao lado do campo no formulário, que ao ser clicado, vai realizar uma consulta em um local definido e retornar uma lista para o usuário, que vai poder escolher a opção desejada. O objeto *zoom* possui duas funções (*init* e *start*).

Init do objeto *zoom* é a função chamada pela função *start* do objeto *fields*, onde devem ser passados os parâmetros: Configuração do Campo e Sufixo do Nome do Campo. O objetivo desta função é verificar as configurações do campo consulta recebidas e sempre

chamar a função *start* do objeto *zoom*, mas, dependendo das configurações, os parâmetros passados para a *start*, vão ser diferentes.

Start do objeto *zoom* é a função chamada pela função *init* do objeto *zoom*, onde devem ser passados os parâmetros: jQuery do Campo, Opções do Campo do Tipo Consulta, Função Executada Depois da Função Zoom, Lista de Campos que Vão Receber Retorno da Função Zoom (se necessário) e Sufixo do Nome do Campo (se necessário). O objetivo desta função é passar as configurações recebidas para uma função que vai ser apresentada na Subseção 3.1.9, que vai aplicar a configuração no campo desejado.

Money é o objeto que contém as funções que atuam nos campos do formulário do tipo monetário. Neste tipo de campo, o conteúdo pode ter um prefixo, uma pontuação para separar os milhares, uma pontuação para separar os decimais e só vai aceitar que o usuário digite caracteres com números. O objeto *money* possui duas funções (*init* e *start*).

Init do objeto *money* é a função chamada pela função *start* do objeto *fields*, onde deve ser passado o parâmetro: Configuração do Campo. O objetivo desta função é verificar as configurações do campo monetário recebidas e sempre chamar a função *start* do objeto *money*, mas, dependendo das configurações, os parâmetros passados para a *start*, vão ser diferentes.

Start do objeto *money* é a função chamada pela função *init* do objeto *money*, onde devem ser passados os parâmetros: jQuery do Campo e Opções do Campo do Tipo Monetário. O objetivo desta função é com as configurações recebidas, aplicar uma máscara no campo desejado, para ser do tipo monetário com um padrão e não deixar que o usuário insira caracteres que não sejam números. Outra vantagem de se criar esta máscara, é que o valor desse tipo de campo, costuma ser utilizado para realizar cálculos durante o processo e se não existir um padrão nos valores dos campos, esses cálculos podem ficar complexos de serem realizados.

Date é o objeto que contém as funções que atuam nos campos do formulário do tipo data. Neste tipo de campo, vai existir um botão ao lado do campo no formulário, que ao ser clicado, vai ser exibido um calendário para o usuário escolher a data desejada, o ideal, é que o desenvolvedor bloqueie este campo no HTML para o usuário nunca poder inserir dados digitando, assim, o único modo dele inserir uma data, vai ser pelo calendário, mantendo um padrão para todos os campos. O objeto *date* possui duas funções (*init* e *start*).

Init do objeto *date* é a função chamada pela função *start* do objeto *fields*, onde deve ser passado o parâmetro: Configuração do Campo. O objetivo desta função é verificar as configurações do campo data recebidas e sempre chamar a função *start* do objeto *date*, mas, dependendo das configurações, os parâmetros passados para a *start*, vão ser diferentes.

Start do objeto *date* é a função chamada pela função *init* do objeto *date*, onde

devem ser passados os parâmetros: jQuery do Campo e Opções do Campo do Tipo Data. O objetivo desta função é com as configurações recebidas, criar a variável calendário padrão da plataforma para o campo e adicionar esta variável na lista de campos que são do tipo data, assim, a própria plataforma vai reconhecer o campo como sendo deste tipo e com as configurações desejadas, vai criar o calendário para o campo.

Aprovação é o objeto que contém as funções que atuam nos campos do formulário do tipo aprovação. Apesar de estar como sendo um campo, na verdade a aprovação envolve um seção inteira, sendo os elementos: Campo do tipo radio, onde o usuário escolhe se foi aprovado, reprovado ou deve ir para ajuste, campo para nome do aprovador, campo para data da aprovação/reprovação, campo para id do aprovador, campo para email do aprovador, campo de observações da aprovação/reprovação e uma área para mensagem de aprovação. O objeto *date* possui duas funções (*init* e *valid*).

Init do objeto *aprovacao* é a função chamada pela função *start* do objeto *fields*, onde deve ser passado o parâmetro: Configuração do Campo. O objetivo desta função é resgatar do HTML cada elemento da seção de aprovação, transformando em elementos jQuery, criar uma função para executar quando houver uma alteração no campo radio e chamar a função *valid*. A função executada na alteração do campo radio, vai atualizar os campos data da aprovação/reprovação, nome do aprovador, id do aprovador e email do aprovador, em seguida, vai chamar a função que cria uma mensagem e exibe as informações inseridas nos campos para o usuário.

Valid do objeto *aprovacao* é a função chamada pela função *init* do objeto *aprovacao*, onde devem ser passados os parâmetros: Configuração do Campo, Variável com Campo Radio da Aprovação e Variável com Campo Observações da Aprovação. O objetivo desta função é criar regras para que o campo radio da aprovação se torne obrigatório e o campo observações da aprovação se torne obrigatório caso seja reprovado.

customActions é o objeto que contém os elementos para aplicar as funções customizadas nos campos. Estas funções customizadas existem quando o usuário quer aplicar mais alguma configuração em um campo que não seja as padrões já feitas pelo *framework*, dando total liberdade para o desenvolvedor criar o que desejar. O objeto *customActions* possui duas funções (*init* e *start*).

Init do objeto *customActions* é a função chamada pela função *start* do objeto *fields*, onde deve ser passado o parâmetro: Configuração do Campo. O objetivo desta função é verificar se foi definida uma função customizada para o campo e chamar a função *start* do objeto *customActions*.

Start do objeto *customActions* é a função chamada pela função *init* do objeto *customActions*, onde devem ser passados os parâmetros: jQuery do Campo e Função Customizada. O objetivo desta função é aplicar a função customizada recebida no campo,

independente do tipo de função que o desenvolvedor criou.

3.1.4 Objeto *sections*

O objeto *sections* foi criado para conter todos os objetos e funções que atuam nas seções do formulário. No momento foi criada apenas uma função chamada *init*.

Init é a função chamada pela função *init* principal do *framework*, onde devem ser passados os parâmetros: Número da Atividade e Configuração das Seções. O objetivo desta função é realizar um *loop* em todas as configurações de seções recebidas e aplicar as mesmas seguindo as regras. Se configuração informa que seção é sempre visível, faz com que seção seja visível, se não, verifica se deve ser visível no número da atividade atual do formulário e se verdadeiro, faz com que seção seja visível. Se configuração informa que seção não é editável, faz com que seção seja desabilitada, se não, verifica se deve ser editável no número da atividade atual do formulário e se falso, faz com que seção seja desabilitada.

3.1.5 Objeto *tables*

O objeto *tables* foi criado para conter todos os objetos e funções que atuam nas tabelas do formulário. Quando se é falado de tabelas, não são as tabelas estáticas que podem existir no formulário, mas sim, as tabelas chamadas de Pai e Filho pelos desenvolvedores. Estas tabelas Pai e Filho possuem um botão para adicionar linha dinamicamente, com o conteúdo definido em uma única linha no HTML, além disso, deve existir na linha um botão para excluir a linha, assim, o usuário pode adicionar e remover linhas destas tabelas. O objeto *tables* contém três funções (*init*, *start*, *initEvents*).

Init é a função chamada pela função *init* principal do *framework*, onde devem ser passados os parâmetros: Número da Atividade, Configuração das Tabelas. O objetivo desta função é realizar um *loop* em todas as configurações de tabelas recebidas, verificar se configuração deve ser aplicada no estado e número da atividade atual do formulário e passando por todas estas condições, chamar a função *initEvents*. Além disso deve verificar se existe campos na configuração da tabela e se existir deve fazer um *loop* nos mesmos, chamando a função *start*.

InitEvents é a função chamada pela função *init* do Objeto *tables*, onde devem ser passados os parâmetros: Configuração da Tabela. O objetivo desta função é procurar o botão de adicionar linha na tabela e criar uma função para ser executada quando este botão for clicado que por sua vez, vai adicionar linha na tabela e criar função para quando botão de excluir linha for clicado.

Start é a função chamada pela função *init* do Objeto *tables*, onde devem ser passados os parâmetros: Nome da Tabela e Configuração do Campo da Tabela. O objetivo desta

função é aplicar a configuração no campo da tabela, que é feito, chamando a função *start* do objeto *fields*, com isto, o código é reaproveitado, já que apesar de estar dentro da tabela, o campo é igual aos outros que estão fora da tabela, a única diferença, é que os campos de cada linha da tabela possuem um sufixo para ser determinado de qual linha ele pertence e manter o nome como chave primária, por este motivo, o sufixo deve ser passado para a função *start* do objeto *fields*.

3.1.6 Objeto *customActions*

O objeto *customActions* foi criado para conter todos os objetos e funções que atuam nas funções customizadas do formulário. Estas funções customizadas são criadas pelo desenvolvedor para realizarem algum tipo de ação no formulário que não estão presentes no *framework*. Como as funções customizadas dos campos do formulário, estas funções existem, para dar liberdade ao desenvolvedor. O objeto *customActions* contém duas funções (*init* e *start*).

Init é a função chamada pela função *init* principal do *framework*, onde devem ser passados os parâmetros: Estado da Atividade, Número da Atividade e Funções Customizadas. O objetivo desta função é realizar um *loop* em todas as funções customizadas recebidas, verificar se função deve ser aplicada no estado e número da atividade atual do formulário e passando por todas estas condições, chamar a função *start*.

Start é a função chamada pela função *init* do Objeto *customActions*, onde deve ser passado o parâmetro: Função Customizada. O objetivo desta função é aplicar a função customizada recebida no formulário, independente do tipo de função que o desenvolvedor criou.

3.1.7 Objeto *defaults*

O objeto *defaults*, foi criado para conter todos os objetos e funções, que vão ser o *default* nos casos que não existir uma configuração definida. Até o momento as funções e objetos existentes são utilizadas quando o usuário não passa para o *framework* a configuração de algum campo. O objeto *defaults* contém duas funções (*zoomReturn* e *zoomFields*) e três objetos (*dateOptions*, *validOptions* e *moneyOptions*).

ZoomReturn é a função utilizada, quando não é passado uma função, para ser executada depois do usuário escolher uma opção em um campo do tipo consulta. Esta função, vai colocar o retorno escolhido pelo usuário, nos campos do formulário com o mesmo nome do retorno. Desta forma, pode ser muito útil, quando os campos do formulário têm o mesmo nome do retorno, pois o desenvolvedor não precisa criar uma função para cada campo do tipo consulta, mas, se os campos não forem do mesmo nome, vai acabar perdendo dados do retorno.

ZoomFields é a função utilizada, quando a configuração do tipo de função que vai ser executada depois do usuário escolher uma opção em um campo do tipo consulta, for 1. Neste tipo de configuração, o usuário vai passar nas configurações, uma lista, informando o campo do formulário que vai ser colocada cada retorno, nestes casos, essa é a função que vai fazer a associação.

DateOptions é o objeto com padrões de configuração, quando não for passado pelo desenvolvedor, as configurações de um campo do tipo data. Até o momento, a única configuração padrão para este tipo de campo, é não usar a data atual.

MoneyOptions é o objeto com padrões de configuração, quando não for passado pelo desenvolvedor, as configurações de um campo do tipo monetário. Três configurações são utilizadas nestes casos, não utilizar nenhum prefixo para o valor do campo, não utilizar nenhuma pontuação para separar os milhares e utilizar vírgula para separar os decimais.

ValidOptions é o objeto com padrões de configuração, quando não for passado pelo desenvolvedor, a dependência para que seja necessária a validação do campo. Nestes casos, a dependência padrão definida, é que a regra de validação só vai ser verificada quando o campo for visível.

3.1.8 Objeto *utils*

O objeto *utils*, foi criado para conter todos os objetos e funções, que podem ser utilizados por várias partes do *framework*. A principal vantagem de ter este objeto, é reaproveitar o código, pois, se o mesmo não fosse utilizado, seria necessário replicar o mesmo trecho de código, em vários outros objetos do *framework*. O objeto *utils* contém uma função (*verificaConteudo*) e dois objetos (*addClass* e *validate*).

VerificaConteudo é a função que recebe como parâmetro um valor qualquer e uma lista de valores, retorna *true* quando o valor se encontra na lista e *false* caso contrário. Está função faz parte deste objeto *utils*, pois, é utilizada em vários trechos de código, para verificar se o estado ou número atual da atividade, estão na lista que configuração deve ser executada.

AddClass é o objeto com as funções (*init* e *start*), necessárias para adicionar classes nos campos do formulário, faz parte deste objeto *addClass*, pois, é chamada por várias outras funções que precisam adicionar classes em campos, para executar alguma ação. *Init* é a função chamada pela função que deseja adicionar a classe, passando o parâmetro: Configuração do Campo. Esta função verifica se existe classes para serem adicionadas e chama a função *start*. A função *start* é chamada pela *init*, passando os parâmetros: jQuery do Campo e Lista de Classes. Seu objetivo é fazer um *loop* na lista de classes e adicionar a classe no campo, utilizando o elemento jQuery.

Validate é o objeto com as funções (*init* e *start*), necessárias para criar regras de

validação nos campos do formulário, da mesma maneira que a função *addClass*, faz parte deste objeto *utils*, pois, é chamada por várias outras funções que precisam adicionar regras de validação nos campos. *Init* é a função chamada pela função que deseja adicionar a regra de validação, passando o parâmetro: Configuração do Campo. Esta função verifica se existe validações para serem criadas para o campo e faz *loop* em todas as validações, chamando a função *start*. A função *start* é chamada pela *init*, passando os parâmetros: Elemento JQuery do Campo, Configuração do Campo e Dependência para Criar a Regra de Validação. Seu objetivo é criar um objeto com as configurações de validação e utilizar este objeto para criar a regra de validação para o campo.

3.1.9 Funções Reaproveitadas

Durante o desenvolvimento do *framework* foi observada a necessidade de utilizar algumas funções já desenvolvidas por outros desenvolvedores, estas funções contribuem para melhor performance do *framework* e economizaram tempo de desenvolvimento. Até o momento foram utilizadas quatro funções, a intenção desta seção, não é explicar como as mesmas funcionam, mas sim, passar uma visão geral do que elas fazem.

A primeira é a *setDisabled*, esta função é utilizada pelo *framework*, quando existe a necessidade de desabilitar campos do formulário, pois a seção deve estar no modo desabilitado para edição. A principal função, é retirar o campo da visão do usuário e apresentar apenas a *label* e o conteúdo do campo.

A segunda é a *fWZoom*, esta função é utilizada nos campos do tipo consulta, facilitando a iteração com o usuário. Existem diversas maneiras de utiliza-la, mas para o *framework*, ela é útil para exibir o resultado da consulta para o usuário, resgatar a opção escolhida pelo mesmo e guardar as informações nos campos adequados.

As duas últimas atuam na validação dos campos, definindo padrões. A *setDefault* é utilizada para aplicar classes nos campos quando é feita a validação, deixando os mesmos, verde ou vermelho, ficando claro para o usuário, os campos que foram preenchidos da maneira errada. Por último, foi utilizada uma função que faz a tradução nas mensagens exibidas pela validação, que por padrão são em inglês e são traduzidas para o português, por padrão da linguagem, esta função não possui um nome.

3.2 Manual de Utilização

No desenvolvimento deste trabalho, foi criado também, um projeto base, para auxiliar os desenvolvedores na utilização do *framework*. Este projeto base, é um arquivo HTML e um JavaScript que devem ser colocados na pasta do projeto, juntamente com o *script* do *framework* e continuar o desenvolvimento, seguindo os exemplos presentes

nos arquivos. Não é o foco aqui explicar o conteúdo do arquivo HTML, mas sim, como continuar o desenvolvimento do *script* JavaScript do projeto base.

Para utilizar o *framework* no projeto, é necessário criar quatro *arrays* de objetos, com chaves e valores nos padrões estabelecidos pelo *framework*, esses objetos são: *fieldsConfig*, *sectionsConfig*, *tablesConfig* e *customActionsConfig*. Depois de criados os objetos, deve ser chamada a função *init* principal do *framework* passando como parâmetros os objetos criados, o estado da atividade (estado considerado pela plataforma, podendo ser: MOD, VIEW ou ADD) e o número atual da atividade (código dado pela plataforma, representado por um número inteiro). O restante do *script* do projeto, pode ser desenvolvido de acordo com as preferências do desenvolvedor.

3.2.1 Array *fieldsConfig*

FieldsConfig, é um *array* onde cada elemento, é um objeto com as configurações de um campo do formulário. Durante o desenvolvimento, deve ser criado um objeto para cada campo que não seja um *input* simples, com as configurações desejadas para o mesmo. Até o momento, o *framework* cria as configurações de quatro tipos de campos (consulta, monetário, data e aprovação). Os atributos que devem ser passados dentro do objeto de cada campo são:

- *name*: *String* com o nome do campo que foi colocado na estrutura HTML, este nome que é utilizado para o *framework* encontrar o campo no formulário e aplicar as configurações, então, é um atributo obrigatório.
- *state*: Objeto com as definições de quando a configuração deve ser aplicada, é um atributo obrigatório pois é utilizado pelo *framework*, para decidir se aplica ou não a configuração naquele momento. Dois atributos devem ser passados dentro deste objeto:
 - *type*: *Array* de *Strings*, com estados da atividade que devem ser aplicadas as configurações, exemplo: ['ADD', 'VIEW']. Pode ser passado uma *string* com valor *default*, ao invés do *array*, assim o *framework* vai considerar ['ADD', 'MOD'].
 - *num*: *Array* de inteiros, com números das atividades que devem ser aplicadas as configurações, exemplo: [1, 3, 4].
- *fieldType*: *String* com tipo do campo, é um atributo obrigatório pois é utilizado pelo *framework* para definir o tipo de configuração que deve ser aplicada. Pode assumir os valores: *zoom* (campos do tipo consulta), *money* (campos do tipo monetário), *date* (campos do tipo data) e *aprovacao* (campos do tipo aprovação).

- *validate*: Array de *Strings*, com os tipos de validação que devem ser aplicadas no campo, não é um atributo obrigatório, pois, não é sempre que vão existir regras de validação. Se não for passado este atributo, o *framework* vai entender que não é necessária nenhuma validação, exemplo: ['required', 'date'].
- *requiredConfig*: Objeto com apenas um atributo que deve chamar *depends*, o valor deste atributo, deve ser uma função que recebe o próprio campo como parâmetro e sempre deve retornar *true* ou *false*. O *framework* só vai aplicar as regras de validação definidas no atributo *validate* quando a função retornar *true*. Não é um atributo obrigatório, pois se não for passado, o *framework* vai sempre aplicar as regras de validação e este atributo só vai ser utilizado se for passado regras de validação no atributo *validate*.
- *class*: Array de *Strings*, com as classes que devem ser colocadas no campo, não é um atributo obrigatório, pois, não é sempre que vai ser necessário incluir classes no campo. Se não for passado este atributo, o *framework* vai entender que não é necessário adicionar classes, exemplo: ['text-right', 'form-control'].
- *fieldOptions*: Objeto com as opções do campo, deve ser passado quando o campo for do tipo data ou monetário, não é um atributo obrigatório, pois quando não passado, o *framework* utiliza as opções padrões para o tipo do campo. Quando o campo for do tipo data, algumas opções de atributos que podem ser adicionados no objeto são: *maxDate*, *minDate* e *useCurrent*. Quando o campo for do tipo monetário, algumas opções são: *prefix*, *thousands* e *decimal*.
- *zoomOptions*: Objeto com as opções do campo, deve ser passado quando o campo for do tipo consulta, é um atributo obrigatório, pois quando não passado, não é possível configurar o campo do tipo consulta. Alguns exemplos das opções que devem ser passadas no objeto são: *label*, *fWZoomType* e *CodQuery*.
- *zoomReturn*: Objeto com as opções de retorno do campo do tipo consulta, é um atributo obrigatório, pois quando não passado, não é possível realizar o tratamento dos dados retornados para os campos do formulário. Os atributos necessários dentro do objeto são:
 - *type*: Inteiro informando o tipo de tratamento do retorno. Quando é passado 1, o *framework* vai colocar os dados retornados, nos campos informados no *array* passado no atributo *fields*. Quando é passado 2, o *framework* executa a função passada no atributo *fields*. Quando é passado a *string default*, o *framework* executa a função padrão, colocando os dados retornados, nos campos do formulário com nomes iguais aos dos dados no retorno da consulta.

- *fields*: Quando é passado 1 no atributo *type*, deve ser um *array* de objetos, informando no atributo *data*, o nome do dado no retorno da consulta e no atributo *formField*, o nome do campo do formulário que deve ser colocado o dado. Quando é passado 2 no atributo *type*, deve ser uma função, que da maneira que o desenvolvedor achar melhor, deve colocar os dados retornados da consulta nos campos do formulário. Quando é passado *default* no atributo *type*, não é necessário passar este atributo.

3.2.2 *Array sectionsConfig*

SectionsConfig, é um *array* onde cada elemento, é um objeto com as configurações de uma seção do formulário. Durante o desenvolvimento, deve ser criado um objeto para cada seção, com as configurações desejadas para a mesma. Até o momento, os atributos que devem ser passados dentro do objeto de cada seção são:

- *id*: *String* com o nome de uma classe, que deve estar presente em uma *tag section* da estrutura HTML. Esta classe, é utilizada pelo *framework*, para encontrar a seção no formulário e aplicar as configurações, então, é um atributo obrigatório.
- *visible*: Booleano, utilizado para o *framework* saber, quando a seção deve ficar visível no formulário, por isso, é um atributo obrigatório. Quando é passado *true*, a seção vai ser visível em todas as atividades. Quando é passado *false*, a seção vai ser visível, apenas nas atividades encontradas no atributo *visibleAtv*.
- *visibleAtv*: *Array* de inteiros, com números das atividades que a seção deve ser visível, exemplo: [1, 3, 4]. É um atributo obrigatório, apenas se o atributo *visible* for passado com valor *false*.
- *enabled*: Booleano, utilizado para o *framework* saber, quando os campos da seção vão ser editáveis, por isso, é um atributo obrigatório. Quando é passado *false*, os campos não vão ser editáveis em nenhuma atividade. Quando é passado *true*, os campos vão ser editáveis, nas atividades encontradas no atributo *enabledAtv*.
- *enabledAtv*: *Array* de inteiros, com números das atividades que os campos da seção devem ser editáveis, exemplo: [1, 3, 4]. É um atributo obrigatório, apenas se o atributo *enabled* for passado com valor *true*.

3.2.3 *Array tablesConfig*

TablesConfig, é um *array* onde cada elemento, é um objeto com as configurações de uma tabela do formulário. Durante o desenvolvimento, deve ser criado um objeto para cada tabela, com as configurações desejadas para a mesma. Até o momento, os atributos que devem ser passados dentro do objeto de cada tabela são:

- *id*: *String* com o nome da tabela que foi colocado na estrutura HTML, este identificador que é utilizado para o *framework* encontrar a tabela no formulário e aplicar as configurações, então, é um atributo obrigatório.
- *state*: Objeto com as definições de quando a configuração deve ser aplicada, é um atributo obrigatório pois é utilizado pelo *framework*, para decidir se aplica ou não a configuração naquele momento. Dois atributos devem ser passados dentro deste objeto:
 - *type*: *Array* de *Strings*, com estados da atividade que devem ser aplicadas as configurações, exemplo: ['ADD', 'VIEW']. Pode ser passado uma *string* com valor *default*, ao invés do *array*, assim o *framework* vai considerar ['ADD', 'MOD'].
 - *num*: *Array* de inteiros, com números das atividades que devem ser aplicadas as configurações, exemplo: [1, 3, 4].
- *fields*: *Array* de objetos, onde, cada elemento, são as configurações de um campo, definido na linha da tabela no HTML. Este *array* de objetos, é criado da mesma forma que o *FieldsConfig*, com os mesmos atributos e regras, isto, pelo fato, de também serem campos do formulário, a única diferença, é que o *framework* vai entender, que em todas as linhas da tabela, vão existir estes campos. Não é um atributo obrigatório, pois não são todas as tabelas que vão ter campos em suas linhas.

3.2.4 *Array customActionsConfig*

CustomActionsConfig, é um *array* onde cada elemento, é um objeto com as configurações de uma função customizada. A intenção destas funções customizadas, é dar liberdade para o desenvolvedor, desenvolver da maneira que ele achar melhor, os trechos de código que não são padrões do *framework* e vão ser o último código a ser executado, quando a página for renderizada. Para melhor performance, o ideal é que em uma mesma função customizada, sejam realizadas várias ações, mas não existe um limite, podendo existir quantas funções forem necessárias no projeto. Os atributos que devem ser passados dentro do objeto de cada função customizada são:

- *state*: Objeto com as definições de quando a função customizada deve ser executada, é um atributo obrigatório, pois é utilizado pelo *framework*, para decidir se executa ou não, a função naquele momento. Dois atributos devem ser passados dentro deste objeto:

- *type*: *Array* de *Strings*, com estados da atividade que a função deve ser executada, exemplo: ['ADD', 'VIEW']. Pode ser passado uma *string* com valor *default*, ao invés do *array*, assim o *framework* vai considerar ['ADD', 'MOD'].
- *num*: *Array* de inteiros, com números das atividades que a função deve ser executada, exemplo: [1, 3, 4].
- *customActions*: *Function*, à ser executada. O trecho de código presente dentro desta função e os padrões que foram utilizados para desenvolver, não são considerados pelo *framework*, apenas é dado o comando, para que seja executado, no momento informado no atributo *state*. É um atributo obrigatório, pois é a própria função que vai ser executada.

4 Resultados

Para demonstrar as melhorias esperadas no desenvolvimento de um projeto, onde é utilizado o *framework*, foi desenvolvido um projeto exemplo básico, utilizando os arquivos base e o *framework*, como explicado na Seção 3.2. Para melhor entendimento, este capítulo foi dividido em duas subseções, a primeira explicando como foi criado o projeto que utiliza o *framework* e a segunda mostrando o projeto criado. A ideia principal desta seção, é mostrar que o *framework* funciona e pode ser utilizado. No entanto não foram feitos testes de usabilidade para aferir a melhoria na qualidade dos programas, mas isto será realizado em trabalhos futuros.

4.1 Projeto Exemplo com *Framework*

O primeiro passo foi criar um novo projeto e construir seu fluxo BPM, mostrado na Figura 8, neste caso, é um diagrama simples, com apenas três atividades:

- Início: Nesta atividade é apresentado um formulário ao usuário, que deverá ser preenchido contendo as informações da solicitação.
- Aprovação: Nesta atividade o usuário vai visualizar as informações e fazer a aprovação/reprovação.
- Fim: Esta atividade é apenas para visualização das informações, depois que o processo for encerrado.

Figura 8 – Fluxo projeto exemplo



Fonte: Produzido pelo autor.

Em seguida, os arquivos base e o *script* do *framework* foram colocados na pasta do projeto. Para este exemplo, não foi necessário editar o arquivo HTML base, pois o projeto exemplo possui a mesma estrutura. Os elementos presentes neste HTML base são:

- Solicitante: Seção com dois campos: Nome do Solicitante e Email do Solicitante.

- Requisição: Seção com três campos: Data Exemplo, Monetário Exemplo e Consulta Exemplo.
- Tabela: Seção com uma tabela pai e filho, onde cada linha possui três campos: Data Exemplo, Monetário Exemplo e Consulta Exemplo.
- Aprovação: Seção com um botão para escolha da aprovação/reprovação e um campo: Observações da Aprovação.
- Rodapé: Seção com um campo, Informações Adicionais.

Os objetos presentes no JavaScript são *fieldsConfig*, *sectionsConfig*, *tablesConfig* e *customActionsConfig*. A descrição destes objetos e seus respectivos elementos de configuração é:

- *fieldsConfig*:
 - Data Exemplo: Elemento com as configurações para o campo com nome DATA-EXEMPLO. Deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo data e este campo deve ser ainda obrigatório. De acordo com as regras, não poderá assumir data maior que data atual e não pode usar data atual como padrão.
 - Monetário Exemplo: Elemento com as configurações para o campo com nome MONETARIOEXEMPLO, deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo monetário e também obrigatório, se o campo Nome do Solicitante for diferente de vazio. Além disso, deve ser adicionada a classe *text-right*, ter prefixo R\$, nenhuma pontuação para separar as centenas e vírgula para separar os decimais.
 - Consulta Exemplo: Elemento com as configurações para o campo com nome CONSULTA, deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo consulta e não obrigatório. Sua utilização é para consultar um *array* e tratamento de retorno do tipo dois, sendo este tratamento de retorno, associado à uma função que coloca o retorno da consulta nos campos CONSULTA e CONSULTACOD.
 - Aprovação: Elemento com as configurações para os campos da seção de aprovação com nome EXEMPLO, deve ser configurado apenas na atividade Aprovação de forma obrigatória.
- *sectionsConfig*:
 - Solicitante: Elemento com as configurações para a seção secSolicitante, deve ser sempre visível e editável apenas na atividade Início.

- Requisição: Elemento com as configurações para a seção `secRequisicao`, deve ser sempre visível e editável apenas na atividade Início.
 - Tabela: Elemento com as configurações para a seção `secTabela`, deve ser sempre visível e editável apenas na atividade Início.
 - Aprovação: Elemento com as configurações para a seção `secAprovacaoEXEMPLO`, deve ser visível apenas nas atividades Aprovação e Fim e editável apenas na atividade Aprovação.
 - Rodapé: Elemento com as configurações para a seção `secRodape`, deve ser sempre visível e editável nas atividades Início, Aprovação e Fim.
- *tablesConfig*:
 - Tabela pai e filho: Elemento com as configurações para a tabela com nome `tblTabela`, deve ser configurado apenas na atividade Início, em estado *default*. Os campos de cada linha da tabela possuem as seguintes configurações:
 - * Data Exemplo Tabela: Elemento com as configurações para o campo com nome `DATAEXEMPLOTABLE`, deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo data e não obrigatório. Não pode assumir data menor que data atual e não se deve usar data atual como padrão. Além disso, ao configurar este elemento deve ser executada uma função customizada que escreve uma mensagem no console.
 - * Monetário Exemplo Tabela: Elemento com as configurações para o campo com nome `MONETARIOEXEMPLOTABLE`, deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo monetário, não obrigatório, adicionada a classe *text-right*, ter prefixo R\$, não possuir nenhuma pontuação para separar as centenas e vírgula para separar os decimais.
 - * Consulta Exemplo Tabela: Elemento com as configurações para o campo com nome `CONSULTATABLE`, deve ser configurado apenas na atividade Início, em estado *default*, para ser do tipo consulta, não obrigatório, consultar um array e tratamento de retorno do tipo um, sendo um *array* informando o nome de cada campo do formulário que o campo do retorno da consulta deve ser colocado.
 - *customActionsConfig*:
 - Função Customizada: Elemento com as configurações para a primeira função customizada, deve ser executada apenas na atividade Aprovação, em estado *default*. Esta função deve apenas escrever uma mensagem no console.

Durante o desenvolvimento deste projeto exemplo utilizando o *framework*, foi possível notar que o desenvolvimento se tornou mais simples e que desenvolver utilizando um padrão, vai proporcionar uma fácil manutenção do código, por qualquer desenvolvedor que tenha conhecimento dos padrões do *framework*. Uma medida utilizada na programação, para calcular a complexidade de desenvolvimento, é o número de linhas do código fonte do *software*, utilizando esta medida, foi possível perceber que desenvolver o projeto base utilizando o *framework*, diminuiu bastante a complexidade, pois, o mesmo projeto sem utilizar a ferramenta, são necessárias mais ou menos oitocentas linhas de código, utilizando o *framework*, são necessárias pouco mais de duzentos e trinta linhas de código.

4.2 Execução do Projeto Exemplo com *Framework*

Depois de todo o desenvolvimento concluído, o projeto foi exportado para a plataforma e esta seção tem a intenção de mostra-lo. O principal objetivo aqui, é mostrar que as regras criadas de uma maneira simplificada utilizando o *framework*, foram aplicadas ao formulário do processo. Para melhor entendimento, cada atividade do processo, vai ser apresentada separadamente.

4.2.1 Atividade Início

- Seções: É possível observar na [Figura 9](#), que as seções Solicitante (A), Requisição (B), Tabela (C) e Rodapé (D) são visíveis e editáveis nesta atividade.

Figura 9 – Formulário na atividade Início

Início

☰ Formulário Exemplo

A Nome do Solicitante Email do Solicitante

B DATA EXEMPLO MONETÁRIO EXEMPLO CONSULTA EXEMPLO

Este campo é obrigatório. E

C

ID	INFORMAÇÕES

D Informações Adicionais

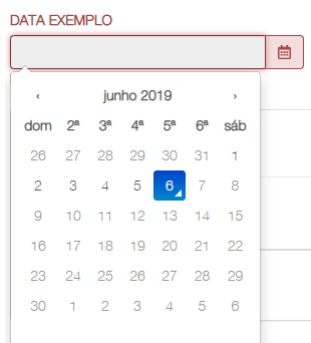
Por favor, utilize a maior quantidade de informações possível.

Fonte: Produzido pelo autor.

- Campos: As [Figura 10](#), [Figura 11](#) e [Figura 12](#) mostram respectivamente os campos Data Exemplo, Monetário Exemplo e Consulta Exemplo, com suas configurações

apresentadas na Seção 4.1. Comparando a Figura 9 (E) com a Figura 13 (A), é possível mostrar uma configuração específica do campo Monetário Exemplo para esta atividade, se tornando obrigatório, quando o campo Nome do Solicitante não é vazio.

Figura 10 – Campo Data Exemplo



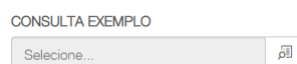
Fonte: Produzido pelo autor.

Figura 11 – Campo Monetário Exemplo



Fonte: Produzido pelo autor.

Figura 12 – Campo Consulta Exemplo



Fonte: Produzido pelo autor.

- Tabelas: Na Figura 14, é mostrado que nesta atividade, existe ação sobre a tabela pai e filho presente no formulário, os campos de cada linha da tabela, também adquirem as configurações apresentadas na Seção 4.1.
- Funções Customizadas: Nenhuma função customizada é executada nesta atividade.

4.2.2 Atividade Aprovação

- Seções: As Figura 15 (A, B e C) e Figura 16 (A e B), mostram que todas as seções são visíveis nesta atividade, mas apenas as seções Aprovação e Rodapé (A e B da

Figura 13 – Regra de obrigatoriedade do campo Consulta Exemplo

Nome do Solicitante: TESTE

Email do Solicitante: [empty]

DATA EXEMPLO: [empty] Este campo é obrigatório.

MONETÁRIO EXEMPLO: [empty] Este campo é obrigatório. **A**

CONSULTA EXEMPLO: Seleccione... [calendar icon]

Fonte: Produzido pelo autor.

Figura 14 – Tabela pai e filho

ID	INFORMAÇÕES
1	DATA EXEMPLO [calendar icon] MONETÁRIO EXEMPLO R\$ 0,00 CONSULTA EXEMPLO Seleccione... [calendar icon]
2	MONETÁRIO EXEMPLO R\$ 0,00 CONSULTA EXEMPLO Seleccione... [calendar icon]
3	MONETÁRIO EXEMPLO R\$ 0,00 CONSULTA EXEMPLO Seleccione... [calendar icon]

+ Item

Fonte: Produzido pelo autor.

Figura 16) são editáveis.

- Campos: Na Figura 17 é possível observar, que os campos presentes na seção Aprovação, adquirem as configurações apresentadas na Seção 4.1 para esta atividade, um dos comportamentos, é mostrado em (A), apresentando uma mensagem quando o usuário seleciona uma das opções do botão de aprovação.
- Tabelas: Nesta atividade não existe ação sobre a tabela pai e filho presente no formulário.
- Funções Customizadas: A Figura 18 apresenta a mensagem exibida no console, referente a função customizada existente para esta atividade.

4.2.3 Atividade Fim

- Seções: Todas as seções são visíveis nesta atividade, mas nenhuma é editável, como mostrado nas Figura 19 (A, B e C) e Figura 20 (A e B).

Figura 15 – Começo do formulário na atividade Aprovação

Aprovação

☰ Formulário Exemplo

A Nome do Solicitante Email do Solicitante

B DATA EXEMPLO MONETÁRIO EXEMPLO CONSULTA EXEMPLO

C

Tabela

ID	INFORMAÇÕES	MONETÁRIO EXEMPLO	CONSULTA EXEMPLO
1	DATA EXEMPLO	R\$ 0,00	CONSULTA EXEMPLO
2	DATA EXEMPLO	R\$ 0,00	CONSULTA EXEMPLO
3	DATA EXEMPLO	R\$ 0,00	CONSULTA EXEMPLO

Fonte: Produzido pelo autor.

Figura 16 – Final do formulário na atividade Aprovação

A Aprovação

Aprovado? Sim Não

Este campo é obrigatório.

Observações da Aprovação

Este campo é obrigatório.

B Informações Adicionais

Por favor, utilize a maior quantidade de informações possível.

Fonte: Produzido pelo autor.

- Campos: Como nenhuma seção é editável nesta atividade, os campos não adquirem nenhuma configuração e ficam apenas visíveis.
- Tabelas: Nesta atividade, não existe ação sobre a tabela pai e filho presente no formulário.
- Funções Customizadas: Nenhuma função customizada é executada nesta atividade.

Figura 17 – Seção Aprovação

Fonte: Produzido pelo autor.

Figura 18 – Ação função customizada

Teste de Execução Função Customizada da Atividade

frameworkBase.js:183

Fonte: Produzido pelo autor.

Figura 19 – Começo do formulário na atividade Fim

ID	INFORMAÇÕES	
1	DATA EXEMPLO	MONETÁRIO EXEMPLO R\$ 0,00
2	DATA EXEMPLO	MONETÁRIO EXEMPLO R\$ 0,00
3	DATA EXEMPLO	MONETÁRIO EXEMPLO R\$ 0,00

Fonte: Produzido pelo autor.

Figura 20 – Final do formulário na atividade Fim

Fonte: Produzido pelo autor.

5 Conclusão

Neste trabalho foi desenvolvido um *framework* para auxiliar no desenvolvimento de um processo com fluxo BPMN, na plataforma Fluig, com o objetivo de gerar padrões e agilidade durante o desenvolvimento. Além disso, foi desenvolvido um projeto teste, que deve servir de base, quando desenvolvedores forem utilizar a ferramenta.

O objetivo proposto, foi alcançado, mas para chegar no resultado final, várias dificuldades foram encontradas. A parte que gerou mais problemas, foi conseguir fazer um código, que fosse ao mesmo tempo, simples, estruturado, performático e funcional, chegando assim, no modelo utilizado, dividindo tudo em objetos.

Para automatizar um processo, utilizando a plataforma Fluig, podem ser utilizados diversos elementos de desenvolvimento web para realizar a ação desejada. Nesta primeira versão da ferramenta, várias funções já são realizadas automaticamente, a maioria para campos e tabelas, passando apenas as configurações desejadas para o *framework*, mas, várias outras ainda podem ser criadas. Como trabalhos futuros, pode ser a criação de qualquer outra função que precisa ser realizada durante a execução de um processo. Um exemplo de função que pode ser implementada no *framework*, seria passar as configurações desejadas para aparecer uma mensagem na tela do usuário, podendo ser estas configurações, o título e conteúdo, além do momento que a mensagem deve aparecer. Em geral, a ferramenta nunca deve chegar em sua versão final, pois sempre vai existir alguma funcionalidade que pode ser implementada.

Referências

CAMARGO, A. H. de et al. Adriano carlos moraes rosa adriano. carlos. rosa@ gmail. com fatec/unifei. Citado na página 17.

COGO, V. V. et al. Análise de apis em javascript para criação de interfaces web ricas. 2009. Citado na página 23.

HABERKORN, E. M. *Gestão empresarial com ERP*. [S.l.]: Microsiga Software, 2003. Citado na página 12.

MOCROSKY, J. F. *Um estudo sobre a aplicação do padrão BPMN (Business process modeland notation) para a modelagem do processo de desenvolvimento de produtos numa empresa de pequeno porte do segmento metal-mecânico*. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2012. Citado na página 18.

OLIVEIRA, J. V. M. d. et al. Desenvolvimento de protótipo de sistema web para gerenciamento de agência lotérica utilizando programação funcional reativa. Florianópolis, SC, 2016. Citado na página 23.

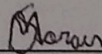
SILVA, A. C. *Business Process Management: introdução sobre BPM em uma visão integrada e didática para a gestão estratégica de processos de negócio*. 1. ed. [S.l.]: Bookess, 2017. Citado 4 vezes nas páginas 14, 15, 18 e 20.

SILVA, M. S. *JavaScript-Guia do Programador: Guia completo das funcionalidades de linguagem JavaScript*. [S.l.]: Novatec Editora, 2010. Citado na página 22.

TERMO DE RESPONSABILIDADE

Eu, Mario Soares de Moraes declaro que o texto do trabalho de conclusão de curso intitulado “*Framework para Desenvolvimento de Processos Baseados na Plataforma FLUIG*” é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 12 de julho de 2019




Mario Soares de Moraes

DECLARAÇÃO DE CONFORMIDADE

Certifico que o(a) aluno(a) **Mario Soares de Moraes**, autor do trabalho de conclusão de curso intitulado “*Framework para Desenvolvimento de Processos Baseados na Plataforma FLUIG*” efetuou as correções sugeridas pela banca examinadora e que estou de acordo com a versão final do trabalho.

João Monlevade, 19 de Setembro de 2019.



Darlan Nunes de Brito