MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS – *Campus* Formiga
Curso de Ciência da Computação

PIPOU – UM SISTEMA DE INFORMAÇÃO PARA O SETOR DE GESTÃO DE PESSOAS DO IFMG – *CAMPUS* FORMIGA

Raí Caetano de Jesus

Orientador: Prof. Me. Diego Mello da Silva

FORMIGA – MG.
2016

RAÍ CAETANO DE JESUS

PIPOU – UM SISTEMA DE INFORMAÇÃO PARA O SETOR DE GESTÃO DE PESSOAS DO IFMG – *CAMPUS* FORMIGA

Trabalho de Conclusão de Curso apresentado ao Instituto Federal *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Diego Mello da Silva

FORMIGA – MG.
2016

J58p Jesus, Raí Caetano de
PIPOU: um sistema de informação para o setor de gestão de pessoas do IFMG – Campus Formiga / Raí Caetano de Jesus. – Formiga, MG., 2016.

140p.: il.

Orientador: Prof. Me. Diego Mello da Silva

Monografia – Instituto Federal Minas Gerais – Campus Formiga.

1. Sistema de informação. 2. Gestão de pessoas. 3. Desenvolvimento web. I. Silva, Diego Mello da. II. Título.

CDD 658.403811

RAÍ CAETANO DE JESUS

PIPOU – UM SISTEMA DE INFORMAÇÃO PARA O SETOR DE GESTÃO DE PESSOAS DO IFMG – *CAMPUS* FORMIGA

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - *Campus* Formiga, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 18 de fevereiro de 2016.

BANCA EXAMINADORA

Prof. Diego Mello da Silva - Orientador

Prof. Mário Luiz Rodrigues Oliveira

Prof. Otávio de Souza Martins Gomes

À Deus,
pelo dom da minha vida , e

À minha noiva e à minha família por terem possibilitado a realização deste sonho.

AGRADECIMENTOS

Primeiramente a minha noiva, por toda compreensão diante às inúmeras vezes que não pude estar presente e por todo seu apoio dedicado a mim para seguir em frente.

Ao meu colega de turma, Roger Santos Ferreira, pelo convite para o desenvolvimento deste projeto e pela confiança em mim depositada.

Ao meu orientador, Prof. Me. Diego Mello da Silva, pelo profissionalismo e competência demonstrados, pelo apoio e constante presença durante o desenvolvimento deste trabalho e por todos os conhecimentos repassados durante o curso.

A todos os professores que contribuíram com tempo e dedicação para uma formação de qualidade.

Aos meus colegas, por todos os momentos compartilhados no decorrer do curso.

“*Andar sobre as águas e desenvolver software a partir de uma especificação é fácil se ambas estiverem congeladas.”*

(Edward V Berard.)

RESUMO

O objetivo desse trabalho é apresentar uma versão inicial de um sistema de informação para o setor de gestão de pessoas do IFMG *campus* Formiga, denominado *Pipou*, fornecendo informações sobre as tecnologias utilizadas, ferramentas de desenvolvimento, modelagem, implementação, interfaces gráficas e funcionalidades implementadas. Esse sistema vem sendo desenvolvido baseado no levantamento de requisitos realizado junto ao responsável pelo setor, a fim de atender às necessidades específicas inerentes aos processos realizados pelo mesmo, melhorando o processo de interação entre o setor e os servidores do instituto, facilitando a execução das demandas solicitadas e a recuperação da documentação sob demanda. O sistema possui o diferencial de ser desenvolvido para a plataforma *web*, o que possibilita o acesso de qualquer lugar e qualquer dispositivo computacional conectado à internet. Foram utilizadas algumas das principais tecnologias para desenvolvimento *web*, como *HyperText Markup Language 5* (HTML5), *Cascading Style Sheets 3* (CSS3), jQuery, Bootstrap, *Asynchronous JavaScript And Xml* (AJAX), *PHP: Hypertext Preprocessor* (PHP) e o *framework* Laravel. Por fim, são apresentadas as considerações finais e um resumo sobre os trabalhos futuros a serem realizados.

Palavras-chaves: Sistema de informação. Gestão de pessoas. Desenvolvimento web.

ABSTRACT

The objective of this work is to present a basic version of an information system named “Pipou”, aimed to help the People Management Department at Instituto Federal de Minas Gerais Campus Formiga, providing information about technologies, development tools, modeling, implementation, graphical interfaces and its features. This system has been developed based on requirements analysis done with information taken from the manager of this department in order to meet the needs inherent of the people management processes. There are expectations that the new system will improve the interaction between the People Management Department and the employees of the Institute, making easy to execute requests and recovery institutional documents. The system was developed for the web platform, enabling its access from anywhere and any device. Were used some of the leading technologies for web development, such as HyperText Markup Language 5 (HTML5), Cascading Style Sheets 3 (CSS3), jQuery, Bootstrap, Asynchronous JavaScript And Xml (AJAX), PHP: Hypertext Preprocessor (PHP) and Laravel Framework. Finally, it will present a suggestion of future works and final remarks about the development of this work.

Keywords: Information system. People management. Web development.

LISTA DE FIGURAS

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS E ABREVIATURAS

AJAX – *Asynchronous JavaScript And Xml*

API – *Application Programming Interface*

ARH – Administração de Recursos Humanos

ASP – *Active Server Pages*

CSRF – *Cross-Site Request Forgery*

CSS – *Cascading Style Sheets*

DER – Diagrama de Entidade Relacional

DOM – *Document Object Model*

DOU – Diário Oficial da União

ERP – *Enterprise Resource Planning*

GB - *Gigabyte*

GP – Gestão de Pessoas

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Enviroment*

JSON – *JavaScript Object Notation*

JSP – *Java Server Pages*

MEC – Ministério da Educação

MVC – *Model-View-Controller*

PHP – *PHP: Hypertext Preprocessor*

PSR – *PHP Standards Recommendation*

RH – Recursos Humanos

SASS – *Syntactically Awesome Style Sheets*

SGML – *Standard Generalized Markup Language*

SGP – Setor de Gestão de Pessoas

SI – Sistema de Informação

SIASS – Subsistema Integrado de Atenção à Saúde do Servidor

TCC – Trabalho de Conclusão de Curso

TCL – *Tool Command Language*

URI – *Uniform Resource Identifier*

URL – *Uniform Resource Locator*

VNI – *Visual Networking Index*

W3C – *World Wide Web Consortion*

WHATWG – *Web Hypertext Application Technology Working Group*

XHTML – *eXtensible HyperText Markup Language*

XHR – *XMLHttpRequest*

XML – *Extensible Markup Language*

XSLT – *Extensible Stylesheet Language Transformations*

SUMÁRIO

1 INTRODUÇÃO

Tradicionalmente, o Setor de Gestão de Pessoas (SGP) é uma das mais importantes áreas de apoio de uma organização e tem um papel crucial no sucesso destas no mercado (RODRIGUES; MORAIS; DA SILVA, 2010, p. 3). Seu principal objetivo é ajudar a organização a atingir seus objetivos e realizar sua missão. Para isso, a Gestão de Pessoas (GP) realiza práticas como agregar talentos à organização, treinar e desenvolvê-los para criar uma organização de aprendizagem, proporcionar excelentes condições de trabalho e melhorar a qualidade de vida no trabalho, avaliar o desempenho humano e melhorá-lo continuamente, modelar o trabalho seja individual ou em equipe a fim de torná-lo significativo, agradável e motivador entre outras. Tudo isso atuando estrategicamente, criando e mantendo um ambiente profissional e positivo na organização (CHIAVENATO, 2010, p. 11-16).

A Gestão de Pessoas (GP) pode ser facilitada pela utilização de um Sistema de Informação (SI) com recursos inteligentes, os quais podem afetar diretamente na tomada de decisões da organização, propondo mudanças nos processos, estrutura e estratégia de negócios, o que pode fortalecer o plano de atuação das organizações, proporcionar a geração de informações rápidas, precisas e úteis, e garantir uma estruturação de gestão diferenciada (BAZZOTTI; GARCIA, 2006). Entretanto, com toda a informatização presente nos tempos de hoje ainda existe carência de sistemas de informação adaptados à GP que permitam o melhor controle das atividades exercidas pelo setor. A gestão de pessoas e seu funcionamento estão ligados diretamente aos processos predominantes na organização, em especial nos setores públicos onde tais processos podem diferir consideravelmente em sua natureza, o que dificulta a escolha de uma solução que se adequa as necessidades particulares do setor.

Além disso, um SGP emite e recebe um grande volume de documentação. A falta de um *software* especializado em gestão documental dificulta e atrasa a recuperação de documentos em repositórios físicos. Tal recuperação pode ser realizada de maneira ágil se o SGP contar com funcionalidades para manutenção de repositório digital de documentos capaz de realizar as pesquisas solicitadas ao setor de forma simples, prática e eficiente. Embora um documento digital não substitua o documento físico em situações que envolvem comprovação de autenticidade e arquivamento, por exemplo, um repositório digital pode conter as versões digitais do documento com informações de localização de sua versão física como numeração de armário e gaveta onde se encontra arquivado tornando mais rápido a recuperação de sua versão física. Nesta situação documentada, um sistema com tais capacidades é capaz de permitir

que o SGP atenda a mais solicitações em um dado intervalo de tempo, agilizando os atendimentos e reduzindo a sobrecarga de trabalho do setor.

A maioria das organizações de médio a grande porte utilizam sistemas de informação conhecidos como *Enterprise Resource Planning*[1] (ERP, Planejamento de Recurso Corporativo), que possuem por objetivo integrar os diversos setores da organização e possibilitar a automação e armazenamento das informações por eles gerado. Um ERP tradicional também contempla módulos para a gestão de pessoas. No entanto, apesar de tais *softwares* proverem funcionalidades comuns à GP (como por exemplo recrutamento e seleção, treinamento, pagamentos, capacitação, auxílios, benefícios entre outras), os mesmos muitas vezes não atendem em sua totalidade às particularidades da organização, sendo preciso personalizá-los, o que aumenta os custos e prazos de implantação. Soma-se a isso o fato de que devido à sua complexidade os ERPs podem não oferecer uma boa usabilidade para seus módulos, e o alto custo para sua aquisição também contribuem para a não adoção de ERPs corporativos em organizações de menor porte ou com orçamento limitado.

Atualmente o IFMG utiliza um ERP corporativo no *campus* Formiga que conta com um módulo para gestão de pessoas. Em um levantamento inicial realizado com o SGP do *campus* sobre as suas funcionalidades identificaram-se alguns pontos importantes sobre o atual sistema, de onde se destacam: (i) o módulo mais utilizado atualmente é o de cadastro de funcionários, visto que o sistema ainda não foi implantado completamente[2], sendo que este módulo colabora apenas com o módulo responsável pelas atividades de Registro Acadêmico do *campus*; (ii) o sistema apresenta um perfil voltado para organizações privadas, disponibilizando em suas interfaces muitos campos que não se aplicam à organizações do setor público (por exemplo, FGTS, horas mensais dentre outros). O excesso de campos fora do contexto do funcionalismo público polui a interface e prejudica a usabilidade do ERP para o usuário do SGP. Além disso, (iii) o atual sistema não possui nenhum recurso capaz de auxiliar a gestão documental de portarias, designações, nomeações, posses e outros artefatos ligados à carreira funcional dos docentes e técnicos administrativos da instituição.

Diante deste cenário, este trabalho tem como objetivos o projeto e desenvolvimento de um sistema de gestão de pessoas denominado "Pipou" adequado para lidar com algumas das necessidades particulares do setor de gestão de pessoas do IFMG *campus* Formiga e capaz de

---

[1] São *softwares* desenvolvidos para controle de vários departamentos e processos de uma organização, integrando-os em um único sistema.

[2] Solicitou-se a customização de alguns módulos do ERP, incrementando novas funcionalidades. Portanto, alguns problemas talvez sejam sanados, no entanto não há previsão para a implantação do produto final.

complementar o atual módulo de gestão de pessoas utilizado no ERP da instituição. Devido à grande quantidade de processos realizados pelo SGP observados durante o levantamento de requisitos, optou-se por considerar como escopo deste projeto a implementação da versão inicial do sistema contendo apenas o módulo de cadastros e um repositório digital de documentos simplificado, porém efetivo, que oferece funcionalidades como criação, alteração e remoção de pastas e organização de documentos em pastas. Ao todo, 19 cadastros foram implementados: Bancos, Cargos, Cidades, Estados, Etnias, Formas de desligamento do servidor, Formas de ingresso do servidor, Memorandos, Nacionalidades, Ofícios, Órgãos expedidores, Órgãos profissionais, Países, Pessoas, Portarias, Regimes de trabalho, Servidores (dados funcionais), Setores e Status do servidor. Com exceção do cadastro de Servidores, implementou-se o restante por completo (lado cliente e lado servidor). Optou-se por implementar somente o lado cliente no cadastro de Servidores uma vez que este influencia diretamente no histórico funcional dos servidores, funcionalidade não contemplada no escopo deste trabalho. Assim, a parte gráfica implementada neste cadastro servirá como base para o desenvolvimento futuro, após a implementação do módulo de históricos funcionais.

Desenvolveu-se tal aplicação para a plataforma *web*, utilizando tecnologias padrão de mercado como: *HyperText Markup Language 5* (HTML5), *Cascading Style Sheets 3* (CSS3), jQuery, *Bootstrap*, *Asynchronous JavaScript And Xml* (AJAX), *PHP: Hypertext Preprocessor* (PHP) e o *framework*[3] Laravel. A evolução das ferramentas para desenvolvimento *web* e as vantagens desta plataforma (adequar-se à inúmeras plataformas de sistemas operacionais e dispositivos, e facilitar a manutenção centralizada) influenciaram na escolha da plataforma *web*. Espera-se com este trabalho que o sistema Pipou auxilie no gerenciamento das atividades realizadas pelo setor, complementando o atual sistema, provendo mais agilidade, segurança e eficiência nos serviços prestados pela Coordenadoria do SGP.

O documento é organizado como segue. A seção 2 apresenta os fundamentos teóricos usados no desenvolvimento deste trabalho, e define conceitos relacionados a gestão de pessoas, sistemas de informação para gestão de pessoas, processo de *software* utilizado, o atual cenário do desenvolvimento *web* e as tecnologias empregadas. Na seção 3 são descritos os materiais e métodos utilizados na execução do projeto, isto é, todos os procedimentos seguidos durante o desenvolvimento da aplicação, como o estudo das tecnologias, o levantamento de requisitos, os esboços de telas elaborados, o ambiente de desenvolvimento e as ferramentas utilizadas, a

[3] Uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica. Pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Mais em: <https://pt.wikipedia.org/wiki/Framework#cite_note-ufcg-1>. Acesso em jan. 2016.

modelagem da aplicação e, por fim, a implementação. A seção 4 mostra os resultados obtidos com o desenvolvimento do sistema Pipou, apresentando suas principais interfaces e funcionalidades aplicadas no módulo de cadastros e no repositório digital de documentos, além de apresentar algumas métricas do sistema. Por fim, a seção 5 apresenta as considerações finais deste trabalho e sugere trabalhos futuros para o sistema Pipou.

2 REFERENCIAL TEÓRICO

2.1 Gestão de pessoas

Uma das áreas que mais são afetadas por mudanças que ocorrem na sociedade é a área de Recursos Humanos (RH). As mudanças são tantas que até o nome da área está sujeito a mudanças. Surgem diferentes nomes para representar um novo espaço e configuração desta. Por exemplo, em muitas organizações, a denominação de Administração/Gestão de recursos humanos (ARH) vem sendo substituída por termos como gestão de talentos humanos, gestão de parceiros ou colaboradores, gestão de competências, gestão do capital humano, administração do capital intelectual e até Gestão de Pessoas ou Gestão com Pessoas (CHIAVENATO, 2010, p. 2).

Segundo Gil (2007), ainda que a expressão Administração de Recursos Humanos seja a mais utilizada, muitas organizações vêm substituindo-a por Gestão de Pessoas devido ao modo de lidar com as pessoas. Para o autor, os argumentos em prol dessa alteração na nomenclatura ressaltam que o termo ARH é de certa forma restritivo, pois implica no entendimento que pessoas que trabalham em uma organização são recursos, tais como recursos materiais e financeiros. Por isso, a literatura adepta à Gestão de Pessoas procura designar pessoas como cooperadores ou parceiros.

Existem muitas definições para a Gestão de Pessoas. Chiavenato (2010, p. 9) define como "a área que constrói talentos por meio de um conjunto integrado de processos e cuida do capital humano das organizações, o elemento fundamental do seu capital intelectual e a base do seu sucesso". Para Gil (2007, p. 17), gestão de pessoas "é a função gerencial que visa à cooperação das pessoas que atuam nas organizações para o alcance dos objetivos tanto organizacionais quanto individuais".

Chiavenato (2010, p. 11-14) afirma que o principal objetivo da GP é ajudar a organização a atingir suas metas, objetivos e realizar sua missão. Por outro lado, o autor destaca que a GP deve contribuir para a eficácia organizacional através de mais alguns objetivos, como: proporcionar competitividade à organização, proporcionar à organização pessoas bem treinadas e bem motivadas, aumentar a auto-atualização e satisfação das pessoas no trabalho, desenvolver e manter qualidade de vida no trabalho, administrar e impulsionar a mudança, manter políticas

éticas e comportamento socialmente responsável, e construir a melhor empresa e a melhor equipe.

Apesar da literatura apresentar conceitos muito abrangentes sobre a área, a Gestão de Pessoas abrange um amplo leque de atividades, as quais estão intimamente relacionadas, o que torna contingencial estabelecer um padrão de classificação capaz de abranger *de facto* todas as atividades, de forma que nenhuma seja excluída, conforme explica Gil (2007). Neste sentido, diversos autores desenvolveram classificações para as atividades de RH, que, de certa forma, considera-se as mesmas para a GP. Chiavenato (2010, p. 15-16) faz essa classificação afirmando que a gestão de pessoas se trata de "um conjunto integrado de processos dinâmicos e interativos", cujo qual consiste de seis processos básicos de GP, sendo estes:

1. **Processos de agregar pessoas**: utilizados para incluir novas pessoas na organização. Incluem recrutamento e seleção de pessoas.
2. **Processos de aplicar pessoas**: utilizados para desenhar as atividades que as pessoas irão realizar, orientar e acompanhar seu desempenho na organização. Incluem desenho organizacional, desenho de cargos, orientação das pessoas e avaliação do desempenho.
3. **Processos de recompensar pessoas**: utilizados para incentivar as pessoas e satisfazer suas necessidades individuais.
4. **Processos de desenvolver pessoas**: utilizados para capacitação e incremento profissional e pessoal das pessoas.
5. **Processos de manter pessoas**: utilizados para criar condições ambientais e psicológicas satisfatórias para as atividades das pessoas.
6. **Processos de monitorar pessoas**: utilizados para acompanhar e controlar as atividades das pessoas e verificar resultados.

Cada um destes processos se relaciona intimamente, de modo que se sobrepõem e se influenciam reciprocamente. Além disso, tendem a favorecer ou prejudicar os demais, se bem ou mal utilizados (CHIAVENATO, 2010). O Quadro 1 resume a moderna Gestão de Pessoas baseada nos seis processos citados.

Quadro 1 - Principais processos da Gestão de Pessoas

Processos	**Atividades**
Suprimento ou Agregação	• Identificação das necessidades de pessoal • Pesquisa de mercado de recursos humanos • Recrutamento e seleção de pessoal
Aplicação	• Análise e descrição de cargos • Planejamento e alocação interna de recursos humanos
Compensação	• Salários • Benefícios e serviços • Carreiras
Desenvolvimento	• Treinamento e desenvolvimento de pessoal • Programas de desenvolvimento e mudança organizacional • Programas de comunicações
Manutenção	• Higiene e segurança no trabalho • Relações com sindicatos • Benefícios
Controle e Monitoramento	• Avaliação de desempenho • Banco de dados • Sistemas de informações gerenciais • Auditoria de recursos humanos

Fonte: Adaptado de (CHIAVENATO, 2010, p.19; GIL, 2007, p. 25)

Contudo, embora exista tal classificação, Chiavenato (2010, p. 8) afirma que a GP

> é uma área muito sensível à mentalidade que predomina nas organizações. Ela é contingencial e situacional, pois depende de vários aspectos, como a cultura que existe em cada organização, da estrutura organizacional adotada, das características do contexto ambiental, do negócio da organização, da tecnologia utilizada, dos processos internos e de uma infinidade de outras variáveis importantes.

Em setores públicos, principalmente, os processos realizados pelo SGP podem diferir consideravelmente. A seção 2.1.1 explica com mais detalhes os processos executados pelo setor de gestão de pessoas do IFMG – *campus* Formiga, que é o *case* usado neste trabalho.

2.1.1 O Setor de Gestão de Pessoas do IFMG – campus Formiga

Em particular, a Coordenadoria do Setor de Gestão de Pessoas do IFMG – *campus* Formiga é responsável por realizar e gerir diversas atividades de apoio ao bom funcionamento da instituição. Em um levantamento prévio realizado com o setor identificou-se atividades que podem ser classificadas como segue:

- **Agendamento de perícias médicas**[4]: utiliza-se o Subsistema Integrado de Atenção à Saúde do Servidor (SIASS) para agendar a perícia médica determinando o dia e horário que o servidor deverá se apresentar. Após, a Coordenadoria do SGP comunica o servidor através de e-mail sobre o agendamento, devendo este comparecer para perícia na data agendada. Feito isso, há a homologação da licença médica pelo médico perito do IFMG que emite um laudo pericial referente aos dias de afastamento do servidor para que a Coordenadoria do SGP tome as devidas providências. Dentro do procedimento, está o arquivamento do laudo no processo de licença médica do servidor.
- **Arquivamento mensal de folhas de ponto**: as folhas de ponto dos servidores são conferidas uma a uma para lançamento de adicional noturno em folha de pagamento. Além disso é realizado a verificação de preenchimento para detectar possíveis inconsistências, rasuras e/ou ausências de assinaturas. Após isso, arquivam-se as folhas de ponto organizadas por mês.
- **Atendimento aos servidores**: várias demandas são encaminhadas via correio eletrônico ao setor, que as resolve. Atualmente não há uma ferramenta capaz de quantificar a produtividade, média de prazo de resposta, solicitações respondidas, solicitações em andamento, entre outras atividades relacionadas à gestão de *workflow*.
- **Atualização de dados cadastrais**: a Coordenadoria do SGP emite anualmente comunicado solicitando que os servidores atualizem seus cadastros e endereços. Para tal, utiliza-se de modelo de formulário eletrônico disponibilizado na página do *campus*, que deve ser preenchido e entregue com comprovante de endereço.
- **Auxílio transporte**: o servidor do *campus* pode requerer a inclusão, alteração ou exclusão de auxílio transporte para o setor. De posse dos documentos comprobatórios originais (comprovante de endereço e passagem rodoviária), a Coordenadoria do SGP realiza a conferência dos documentos, protocola a data de recebimento dos mesmos e inicia processo com a documentação apresentada. Em seguida encaminha para a Pró-Reitoria de Administração do IFMG para análise e registro em folha de pagamento.
- **Cadastro de afastamentos**: quando ocorre, cada tipo de pedido de afastamento gera um processo no qual a Coordenadoria do SGP recolhe a documentação necessária,

[4] Atestados acima de 5 dias e que dentro de um ano somarem mais de 14 dias deverão passar por perícia.

analisa e propõe a homologação do afastamento à autoridade competente, que defere ou indefere o pedido. No caso de deferimento, ocorre a emissão do afastamento e sua publicação em portaria.

- **Cadastro de licenças médicas**: ao receber o atestado médico do servidor, a Coordenadoria do SGP verifica a data de emissão, período de afastamento, Código Internacional de Doenças (CID), assinatura do médico e registra no sistema SIASS a licença médica. Este cadastro ocorre com licenças de até cinco dias quando se trata da saúde pessoal do servidor e de até três dias para acompanhamento de familiar.
- **Cadastro de portarias no histórico funcional**: toda portaria que relaciona um servidor tem sua cópia física armazenada na pasta funcional do servidor.
- **Capacitação de servidores e editais de apoio financeiro**: anualmente se realiza o planejamento de ações de capacitação de servidores junto às diretorias sistêmicas do *campus*. Quando a Coordenadoria do SGP começa a executar as ações de capacitação é necessário realizar controle de gastos com diárias, passagens e inscrições, que são informações base para preenchimento de formulários específicos de capacitação do Ministério da Educação (MEC), fato este que ocorre ao término de cada ano. Este registro também é importante para mensurar o investimento no desenvolvimento dos servidores e estimular o compartilhamento de conhecimentos adquiridos.
- **Contratação de professores substitutos/temporários**: a Coordenadoria do SGP participa da elaboração do edital do processo seletivo simplificado para as vagas de professores temporários e substitutos. Após homologação do resultado final do processo seletivo e publicação do resultado no Diário Oficial da União (DOU), o setor convoca os candidatos aprovados, confere e analisa documentação submetida e, caso completa, contrata o professor selecionado por prazo pré-determinado que poderá ser estendido por até 2 anos, conforme a lei 8.745/1993.
- **Desligamento de servidores**: ocorre a instrução de processo a fim de registrar solicitações de vacância de cargo, exonerações, términos e rescisões contratuais. Posteriormente, registra-se o desligamento do servidor, sendo o desligamento registrado em planilha do Excel e na pasta funcional do mesmo.
- **Emissão de folhas de ponto no final do mês**: no primeiro dia útil do mês, são emitidas as folhas de ponto de todos os servidores do *campus*. A Coordenadoria do

SGP imprime e distribui as folhas de pontos em pastas disponíveis nos setores, a fim de que o servidor faça o registro diário de seu horário de trabalho.

- **Fechamento de malotes às terças e quintas feiras**: todos os requerimentos dos servidores são organizados e encaminhados para a Pró-Reitoria de Administração na Reitoria via malote. Neste momento há emissão de memorando para registrar o encaminhamento de processos.
- **Incentivo a qualificação/retribuição por titulação**: quando o servidor encerra sua qualificação, ele faz requerimento de retribuição por titulação e anexa original e cópia dos certificados/diplomas de cursos de educação formal. A Coordenadoria do SGP instrui processo e encaminha à Pró-Reitoria de Administração para que seja providenciado a análise do processo, publicação de portaria e pagamento do servidor.
- **Ingresso do servidor**: o servidor ingressa no IFMG por meio de concurso público ou processo seletivo simplificado. No caso de concurso público, o candidato aprovado é nomeado, toma posse do cargo e entra em exercício na Instituição onde começa a desempenhar suas atividades. No caso de processos seletivos, o candidato aprovado é contratado por tempo determinado. Ambas as situações exigem que a Coordenadoria do SGP registre a data de ingresso do servidor no Órgão, bem como faça os procedimentos para inclusão em folha de pagamento, enviando processos que tratam de admissão, inclusão de benefícios ou processos que visam registros de dados cadastrais e homologação de solicitações. Por meio desses procedimentos dá-se início a vida funcional do servidor na Instituição, que deverá ser acompanhada pela Coordenadoria do SGP através de registros no assentamento funcional do mesmo, acompanhando-o em todos os acontecimentos decorrentes do seu exercício no Órgão até o seu desligamento.
- **Instrução de processos diversos**: cada direito e dever associado ao servidor pode gerar uma instrução de processo, no qual a Coordenadoria do SGP atribui um número, capa contendo nome, assunto e fundamento legal, legislação impressa, requerimento e documentos comprobatórios, memorando de encaminhamento da Coordenadoria do SGP para a Pró-Reitoria de Administração do IFMG, numeração de página e rubrica em todas as folhas do processo. Feito isso, envia-se o processo para a Pró-Reitoria de Administração para análise e providências.

- **Progressões funcionais por mérito profissional e desempenho acadêmico**: a Coordenadoria do SGP monitora as datas de progressão funcional dos servidores e encaminha os processos para que a chefia imediata, juntamente com o servidor, possa realizar a avaliação da referida progressão. Após isso, a Coordenadoria do SGP providencia minuta de portaria e envia a Pró-Reitoria de Administração do IFMG para que se providencie a publicação de portaria e pagamento do servidor.
- **Progressão por capacitação**: o servidor realiza cursos de capacitação e, ao final, entrega original e cópia dos certificados junto com o requerimento de progressão. A Coordenadoria do SGP instrui processo e analisa se o curso está relacionado ao ambiente organizacional em que o servidor está lotado. Após análise, em caso de deferimento, emite-se minuta de portaria, que é encaminhada à Pró-Reitoria de Administração do IFMG para que se providencie a publicação de portaria e pagamento do servidor.
- **Programação/alteração de férias dos servidores**: anualmente a Coordenadoria do SGP realiza programação de férias de todos os servidores. Estes preenchem um formulário e o entregam a Coordenadoria devidamente assinado pela chefia imediata e servidor. Alteração das férias pode ser realizada conforme demanda.
- **Ressarcimento de mensalidades do plano de saúde**: mensalmente, o setor recebe originais e comprovantes de pagamento do plano, realiza a conferência de autenticidade, solicita o ressarcimento em sistema próprio da Reitoria (“SirSaúde”) e arquiva a documentação na pasta funcional do servidor.
- **Substituição remunerada**: quando há substituição de servidor com função gratificada ou cargo de direção, o servidor substituto solicita à Coordenadoria do SGP substituição remunerada no período em que realizou a substituição. A Coordenadoria confere o requerimento e anexos, assina o pedido, instrui processo e o encaminha à Pró-Reitoria de Administração para pagamento.

O volume de documentos gerado pelo SGP na execução das atividades listadas acima é grande. Toda atividade gera algum tipo de documento, desde um simples comprovante até uma emissão de portaria, memorando, dentre outros. Em se tratando do âmbito público, toda essa documentação produzida deve ser arquivada, não podendo ser descartada antes do prazo previsto por lei. Segundo a Coordenadoria do SGP do IFMG, o setor recebe e emite entre 300 a 400 documentos por mês, envolvendo originais e cópias, sendo que há meses em que esse

número varia devido a demanda de serviço. Isto posto, o volume total de documentos gerados é de difícil manipulação.

Diante do exposto, torna-se claro que é preciso buscar formas de automatizar a execução do *workflow* do SGP a fim de tornar o atendimento das demandas mais ágil e eficiente. Tal automatização pode ser alcançada com uso de sistemas de informação específicos para o SGP. A principal dificuldade enfrentada pelo SGP do *campus* Formiga é a falta de um sistema de informação capaz de atender completamente as necessidades particulares do setor e *campus*. Atualmente, o controle de algumas das atividades documentadas acima é realizado em planilhas do Excel, que não são adequadas para essa gestão pois aumentam de tamanho com o tempo, tornando difícil seu entendimento e gerenciamento, prejudicando os serviços do setor. Para outras atividades não existe controle por planilhas, sendo preciso recorrer a arquivos físicos toda vez que a recuperação de uma informação se torna necessária.

2.2 Sistemas de informação para a gestão de pessoas

Em consequência aos constantes avanços tecnológicos e científicos, a sociedade tem se transformado a cada dia. No cenário atual, altamente competitivo, tais transformações ocorrem cada vez com mais frequência, fazendo com que as organizações se atualizem em virtude às necessidades e urgências impostas pelo meio em que estão inseridas (TELES; DE AMORIM, 2013).

Neste cenário, o desafio da gestão de pessoas tem sido acompanhar a evolução na forma como se administram as organizações em uma economia globalizada num mundo ligado pela tecnologia da comunicação. Como consequência, a gestão de pessoas tem se modificado constantemente. Se a pouco tempo o foco dos gestores de pessoas estava em realizar atividades burocráticas e de controle, atualmente, a forma como se gerencia as pessoas passou a ser um diferencial estratégico independentemente do porte ou nacionalidade da organização (TEGON, 2006).

O setor de gestão de pessoas não se preocupa mais somente com a área operacional, tendo atuado em funções mais estratégicas dentro da organização (CHIAVENATO, 2010, p. 530; RODRIGUES; MORAIS; DA SILVA, 2010, p. 4). De maneira geral, a GP é a função que permite a colaboração eficaz das pessoas (empregados, funcionários, recursos humanos, talentos ou qualquer outra denominação que seja utilizada) para alcançar os objetivos

organizacionais e individuais (CHIAVENATO, 2010, p. 11). Para cumprir com esse novo papel, faz-se uso de sistemas de informação para gestão de pessoas que permitam aos gestores de pessoas obter informações para tomada de decisão, para propor mudanças nos processos, estrutura e estratégia de negócios, servindo assim como apoio no cumprimento dos objetivos organizacionais.

Laudon e Laudon (2007, p. 9) definem tecnicamente um SI como sendo "um conjunto de componentes inter-relacionados que coletam (ou recuperam), processam, armazenam e distribuem informações destinadas a apoiar a tomada de decisões, a coordenação e o controle de uma organização". Neste sentido, Chiavenato (2010, p. 507) conceitua especificamente um sistema de informação para gestão de pessoas como "um sistema utilizado para coletar, registrar, armazenar, analisar e recuperar dados a respeito dos recursos humanos da organização".

Um sistema de gestão de pessoas é projetado para registro e rastreamento de dados e informação de todos os colaboradores da organização e atividades relacionadas a todos os processos da administração de pessoal, interligando todas as suas divisões: recrutamento e seleção, treinamento, pagamentos, capacitação, auxílios, benefícios e demais funções envolvidas no setor. Facilita o relacionamento entre chefias, funcionários e a área de RH, mantendo a transparência e agilidade necessárias para a execução e administração das atividades, uma vez que todo o desenvolvimento dos processos envolvem diversos aspectos, desde a cultura organizacional, até fatores externos como legislação e políticas governamentais. Além do mais, tal sistema pode ser flexível e adaptável, atendendo às necessidades específicas de cada organização (AÇÃO SISTEMAS, 2012).

Em um sistema de informação para gestão de pessoas existem dois objetivos básicos: reduzir custos e tempo de processamento da informação; e proporcionar suporte para decisão, isto é, auxiliar gerentes e colaboradores a tomar decisões eficazes (CHIAVENATO, 2010, p. 507). Sob o ponto de vista organizacional destacam-se outros objetivos, tais como (BERNARDES; DELATORRE, 2008, p. 8): assegurar o pleno exercício da cidadania; agilizar o acesso aos arquivos e às informações; promover a transparência das ações administrativas; garantir economia, eficiência e eficácia na administração pública ou privada; agilizar o processo decisório; e incentivar o trabalho multidisciplinar e em equipe. Em meio aos benefícios apresentados e cumprimento dos objetivos propostos pelos sistemas de informação à gestão de pessoas, Bazzotti e Garcia (2006) e Ação Sistemas (2012) destacam as seguintes vantagens: economia de custos e otimização de tempo; execução das atividades facilitada e organizada; fortalecimento do plano de atuação das organizações; geração de informações rápidas, precisas

e úteis; garantia de uma estruturação de gestão diferenciada; transparência na gestão; e suporte à tomada de decisões.

Todo sistema de gestão de pessoas deve cobrir determinados aspectos. Segundo Chiavenato (2010, p. 508) alguns dos principais aspectos são:

- Total alinhamento com o planejamento estratégico de gestão de pessoas como base informacional para que ele possa ser implementado com sucesso.
- Formulação de objetivos e programas de ação e práticas de GP.
- Registros e controles de pessoal para efeito operacional de folha de pagamento, administração de férias, faltas e atrasos etc.
- Relatórios sobre remuneração, incentivos salariais, benefícios, dados de recrutamento e seleção, plano de carreiras, treinamento e desenvolvimento, higiene e segurança do trabalho, área médica, como base para o processo decisório gerencial.
- Relatórios sobre cargos e seções, custos envolvidos, análises e comparações.
- Um banco de dados de talentos (internos ou externos) e de competências disponíveis ou necessárias para o sucesso organizacional.
- Outras informações gerenciais relevantes para a organização e para as pessoas.
- Assuntos de interesse profissional dos colaboradores, como oportunidades de promoção, ofertas de carreira, novidades sobre a organização, produtos e serviços, clientes, fornecedores.
- Assuntos de interesse pessoal dos colaboradores, como situação de férias, dados sobre benefícios, auxílios, remuneração etc.

Isto posto, é impraticável imaginar um SGP moderno não informatizado. Com o avanço no desenvolvimento de aplicações, funcionalidades específicas de cada um dos sistemas estão sendo integrada em um único SI denominado *Enterprise Resource Planning* (ERP). Tal sistema abrange todos os setores da organização, incluindo todos os níveis administrativos.

Embora completos, os ERPs possuem altos custos associados com sua aquisição e podem ser genéricos. Normalmente, ERPs são desenvolvidos para o setor privado, o que dificulta o uso imediato para setor público, implicando em custos de customização. Além dos custos, a customização de um ERP pode demorar, impactando no funcionamento da organização caso esta não utilize sistemas ‘a parte’ (DE SOUZA, 2005).

O SPG do *campus* Formiga enfrenta este problema. O setor conta com um módulo do ERP Corporativo do IFMG como suporte para o cumprimento mais rápido, seguro e eficaz de

suas tarefas. Contudo, existem necessidades específicas da Gestão de Pessoas que o atual sistema não atende completamente, pois foi desenvolvido pelo seu fornecedor com ênfase no setor privado. É preciso customizá-lo para adequar-se aos processos do SGP, que implica em altos custos e prazos longos, obrigando o setor no atual momento a utilizar ferramentas menos eficientes para realizar seu trabalho.

Chiavenato (2010) comenta que o primeiro passo para desenvolver um sistema de informação de GP é conhecer quais são as necessidades de informações e como montar o banco de dados de gestão de pessoas para supri-las. Em resumo, deve-se discriminar todos os processos realizados pelo setor (isto é, levantar os requisitos da aplicação que dará suporte ao setor) e então modelar sua base de dados. Para evitar problemas durante o desenvolvimento do SI, sua implementação deve ser feita sistematicamente, seguindo algum processo de *software* consolidado. As próximas seções apresentarão os conceitos principais sobre o processo de desenvolvimento de *software*.

2.3 O processo de *software*

Para um *software* ser bem-sucedido, é necessário aplicar técnicas de Engenharia de *Software*, como um processo adaptável e ágil que conduza a um resultado de alta qualidade, atendendo assim às exigências daqueles que o usarão (PRESSMAN, 2011). Apesar de existirem inúmeros processos de *softwares* diferentes, Sommerville (2011, p. 43) afirma que todos devem incluir quatro atividades fundamentais:

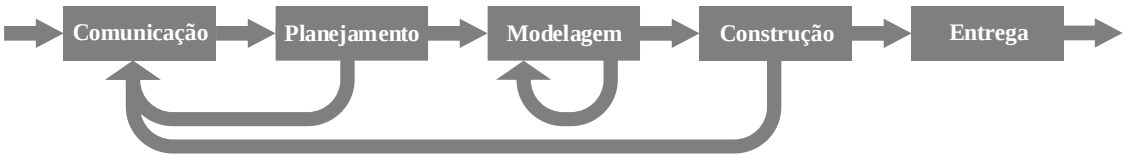
- **Especificação de *software***: as funcionalidades e restrições quanto ao seu funcionamento devem ser definidas.
- **Projeto e implementação de *software***: deve ser produzido para atender às especificações.
- **Validação de *software***: deve ser avaliado para garantir que atenda às demandas do cliente.
- **Evolução de *software***: deve evoluir a fim de atender às necessidades de mudanças do cliente.

Para entender o funcionamento dos processos de *software*, antes é preciso definir um aspecto importante presente nos mesmos, chamado *fluxo de processo*, que, segundo Pressman (2011, p. 54), "descreve como são organizadas as atividades metodológicas, bem como as ações e tarefas que ocorrem dentro de cada atividade em relação à sequência e ao tempo". O autor classifica os fluxos de processo em:

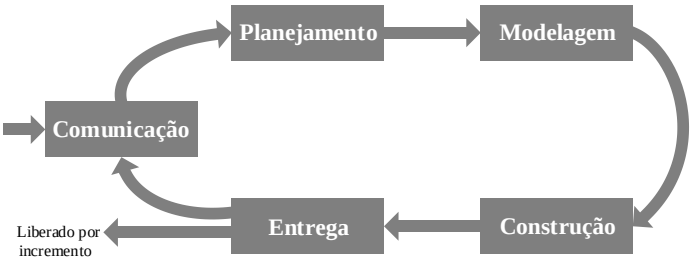
- **Linear**: executa cada uma das atividades em sequência (FIGURA 1.a).
- **Iterativo**: repete uma ou mais atividades antes de prosseguir para a seguinte (FIGURA 1.b).
- **Evolucionário**: executa as atividades de forma circular. Cada volta pelas mesmas conduz a uma versão mais completa do sistema (FIGURA 1.c).
- **Paralelo**: executa uma ou mais atividades em paralelo com outras (por exemplo, a modelagem de um aspecto do sistema poderia ser executa em paralelo com a construção de outro aspecto do sistema) (FIGURA 1.d).

Figura 1 - Fluxos de processo

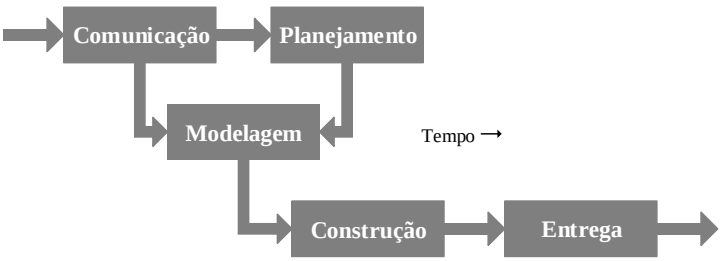
a) Fluxo de processo linear



b) Fluxo de processo iterativo



c) Fluxo de processo evolucionário



d) Fluxo de processo paralelo

Fonte: Adaptado de (PRESSMAN, 2011, p. 54)

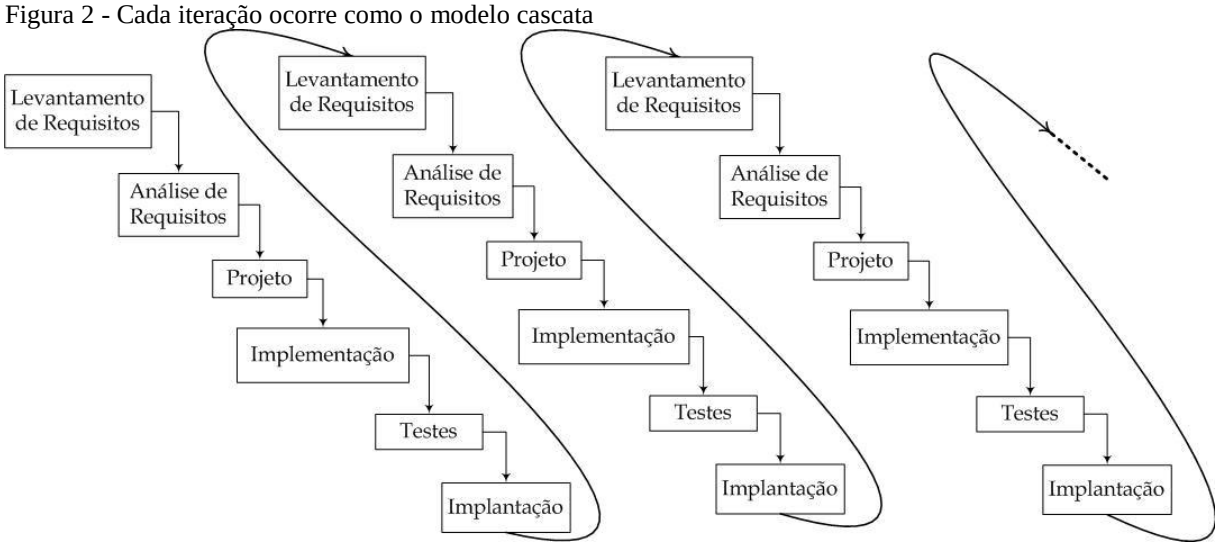
Cada processo de *software* baseia-se em algum destes fluxos. Dentre os modelos mais tradicionais, destacam-se o Modelo Cascata (HIRAMA, 2011; PFLEEGER, 2004; PRESSMAN, 2011; SOMMERVILLE, 2011; WAZLAWICK, 2013), Modelo de Desenvolvimento em Fases (Iterações e Incrementos) (ALMEIDA, 2015; PFLEEGER, 2004; SOMMERVILLE, 2011), Modelos Evolucionários (Prototipação e Espiral) (BOEHM, 1988, 2000; HIRAMA, 2011; PFLEEGER, 2004; PRESSMAN, 2011; SOMMERVILLE, 2011; WAZLAWICK, 2013), e Modelo de Desenvolvimento Iterativo e Incremental.

No caso específico deste trabalho, devido a metodologia aplicada pelo modelo, optou-se pela abordagem de Desenvolvimento Iterativo e Incremental como processo de *software* para o desenvolvimento do sistema Pipou. A seção 2.3.1 apresentará mais detalhes sobre este processo de desenvolvimento.

2.3.1 Modelo de desenvolvimento Iterativo e Incremental

O modelo de desenvolvimento Iterativo e Incremental foi proposto como uma solução aos problemas encontrados no modelo em Cascata. Ele combina a natureza de desenvolvimento iterativa com a incremental, isto é, todo o sistema é desenvolvido em diversos passos similares (iterações), e em cada um destes o sistema é complementado com novas funcionalidades (processo incremental). Em cada uma das iterações (ou ciclos) do desenvolvimento, pode-se identificar fases de Análise, Projeto, Implementação e Testes, o que difere da abordagem clássica, na qual tais fases são realizadas uma única vez. Assim, o desenvolvimento evolui a cada versão, por meio da construção incremental e iterativa de novas funcionalidades até que todo o sistema esteja desenvolvido (BEZERRA, 2015). Na realidade, o modelo Iterativo e Incremental pode ser visto como uma generalização da abordagem em Cascata, onde cada incremento é desenvolvido em cascata, como ilustra a Figura 2. Nela, em cada incremento são realizadas as atividades de Levantamento de Requisitos, Análise de Requisitos, Projeto, Implementação, Testes e Implantação do objeto da iteração, repetindo-se até que todo o sistema seja finalizado.

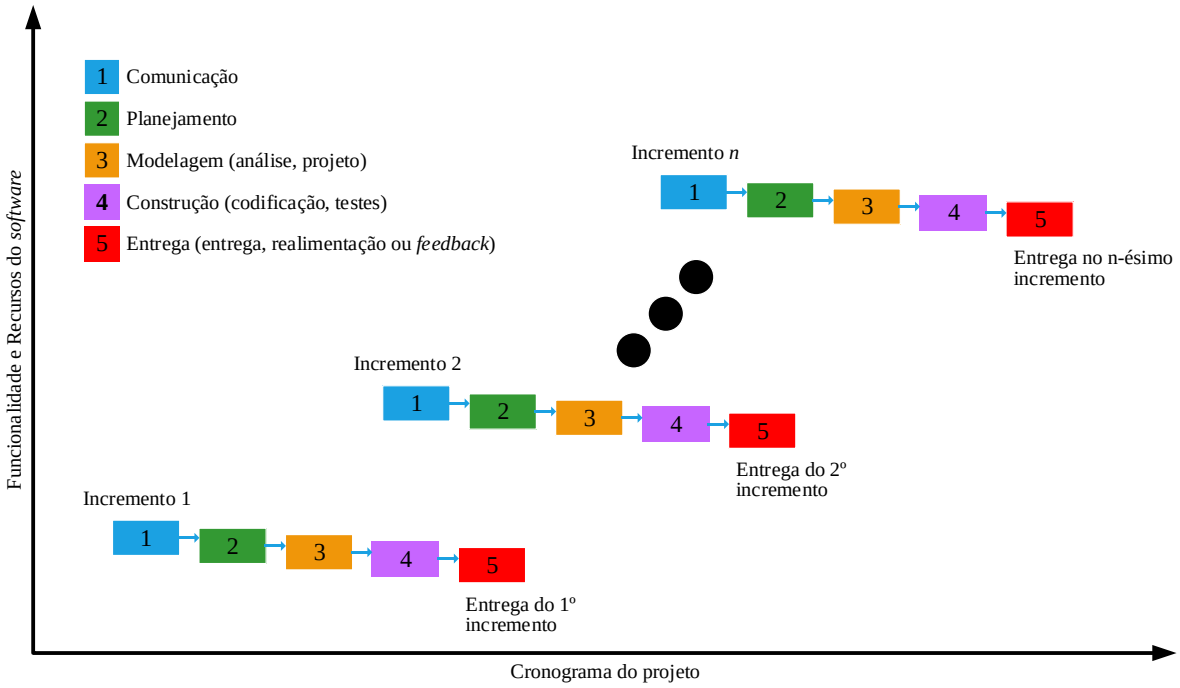
Figura 2 - Cada iteração ocorre como o modelo cascata



Fonte: (BEZERRA, 2015, p. 33)

Este modelo combina os fluxos de processo linear, iterativo e paralelo (FIGURA 1), ou seja, são aplicadas sequências lineares de forma escalonada à medida que o tempo vai avançando. Cada sequência linear gera um incremento (PRESSMAN, 2011), como mostra a Figura 3.

Figura 3 - Modelo de Desenvolvimento Iterativo e Incremental



Fonte: Adaptado de (PRESSMAN, 2011, p. 61)

Bezerra (2015) argumenta que o modelo Iterativo e Incremental deve ser aplicado somente se existir um mecanismo para dividir os requisitos do sistema em partes, de modo que cada uma seja associada a um ciclo de desenvolvimento. O autor ainda destaca que os fatores

considerados nessa divisão são a prioridade (i.e., a relevância do requisito para o cliente) e o risco de cada requisito. Isto posto, é possível elencar algumas vantagens e desvantagens dessa abordagem (BEZERRA, 2015).

Vantagens:

- Redução dos riscos envolvendo custos a um único incremento. Se os desenvolvedores precisarem repetir a iteração, a organização perde somente o esforço mal direcionado de uma iteração, e não o valor de um produto inteiro.
- Redução do risco de lançar o projeto no mercado fora da data planejada. Identificando os riscos numa fase inicial, o esforço despendido para gerenciá-los ocorre cedo, quando as pessoas estão sob menos pressão do que numa fase final de projeto.
- Aceleração do tempo de desenvolvimento do projeto como um todo, porque os desenvolvedores trabalham de maneira mais eficiente quando buscam resultados de escopo pequeno e claro.
- Reconhecimento de uma realidade frequentemente ignorada: as necessidades dos usuários e os requisitos correspondentes podem não estar totalmente definidos no início do processo. Eles são tipicamente refinados em sucessivas iterações. Este modelo de operação facilita a adaptação a mudanças de requisitos.

Desvantagens:

- Dificuldade de gerenciamento devido a possibilidade de ocorrência simultânea das fases do ciclo.
- O usuário pode criar grande expectativa com a primeira versão do sistema, fazendo-se crer que tal versão já corresponde ao sistema como um todo.
- Como todo modelo, está sujeito a riscos de projeto, tais como: o projeto pode não satisfazer aos requisitos do usuário; a verba do projeto pode acabar; o sistema pode não ser adaptável, manutenível ou extensível; e o sistema pode ser entregue ao usuário tarde demais.

Dentre todos os modelos de processos de *software* existentes, deve-se utilizar aquele que se adequa melhor ao desenvolvimento do sistema considerando restrições e características particulares do projeto, independentemente da plataforma a qual o mesmo é desenvolvido (*web*,

desktop, *mobile*). No caso específico deste projeto optou-se por este modelo visto que iterações foram usadas para evoluir o *software* desde seu conceito inicial até a presente implementação.

Baseando-se neste modelo Iterativo e Incremental optou-se por desenvolver o sistema Pipou em plataforma *web*. O evoluído cenário das tecnologias para o desenvolvimento *web* e suas vantagens que se fazem fundamentais para o ciclo de vida da aplicação caso a aplicação Pipou se popularize e seja usada por SGPs de outros *campus* e instituições foram determinantes para a escolha da tecnologia *web*. A seção 2.4 dá mais detalhes sobre o cenário atual do desenvolvimento *web* e as tecnologias utilizadas na implementação deste trabalho.

2.4 Desenvolvimento *Web*

O desenvolvimento para plataforma *web* vêm crescendo rapidamente, tornando-se uma tendência entre as aplicações. Pode-se dizer que tal fato se dá devido a algumas vantagens proporcionadas pela plataforma, na qual Marcoratti ([200-?]) destaca: ambiente multi-plataforma; é acessível através de navegador de qualquer localidade com acesso à internet; desenvolvimento, manutenção e atualização centralizada; exportação de dados entre usuários remotos usando o protocolo *HyperText Transfer Protocol* (HTTP) é mais fácil do que usar outro protocolo; existe escalabilidade no processamento - havendo necessidade de aumento no poder de processamento, basta fazê-lo no servidor. Por outro lado, vale destacar que desenvolver aplicações para a plataforma *web* também tem suas desvantagens, dentre as quais Marcoratti ([200-?]) cita: necessidade de dominar diversas linguagens (por exemplo, HTML, CSS, JavaScript e PHP descritas na seção 2.4.1); falta de padronização entre navegadores quanto à apresentação de interfaces em HTML; desempenho inferior se comparado às aplicações *desktop* locais, considerando ainda o *overhead* de comunicação entre aplicação cliente e o servidor, e protocolos de comunicação; criar páginas dinâmicas e formulários para entrada de dados é mais trabalhoso que em aplicações *desktop* (produtividade é baixa se comparada ao desenvolvimento *desktop*); interface HTML não é rica em controles gráficos e peca no quesito posicionamento podendo comprometer a elegância do visual da aplicação.

Diversas das desvantagens da plataforma *web* em relação à plataforma *desktop* apontadas pelo autor têm sido minimizadas ou sanadas com a evolução do HTML e CSS, bem como com o surgimento de *Integrated Development Environments* (IDEs, Ambientes de

Desenvolvimento Integrado)[5], editores de texto avançados e diversos *frameworks* para controle e lógica de negócio tanto do lado cliente como do servidor (SOUZA 2010; FRANCO, 2011; NASCIMENTO, 2013).

Diante o aumento expressivo no número de pessoas com acesso à internet, crescendo de 400 milhões no ano 2000 para 3,2 bilhões em 2015[6], diversas tecnologias para o desenvolvimento *web* surgiram. Em sua maioria gratuitas, tais tecnologias têm evoluído consideravelmente tornando o trabalho de codificação *web* produtivo e prazeroso, além de melhorar expressivamente a qualidade e beleza do produto final. As próximas subseções abordarão de maneira sucinta tecnologias utilizadas no desenvolvimento de *sites* e aplicações *web* que foram usadas na implementação deste trabalho.

2.4.1 HTML 5

Tim Berners-Lee, inventor da *web* e atual diretor do *World Wide Web Consortium* (W3C[7]) criou em 1990 um protótipo de navegador que pudesse rodar em computadores da NeXT (empresa de computadores fundada em 1985 por Steve Jobs). Tal protótipo ficou conhecido como *WorldWideWeb* mas foi posteriormente renomeado para Nexus, a fim de evitar confusão com o termo *World Wide Web* criado para definir a *web* (SILVA, 2014a, b). Segundo Silva (2014a), Tim Berners-Lee acreditava ser possível interligar hipertextos em computadores diferentes com uso de *links* globais, também chamados de *hiperlinks*[8]. Ele desenvolveu um sistema próprio e protocolo para recuperar hipertextos, denominado *HyperText Transfer Protocol* (HTTP, Protocolo de Transferência de Hipertexto) e para sua utilização um formato padrão de texto denominado *HyperText Markup Language* (Linguagem de Marcação de Hipertexto) ou simplesmente HTML, o qual foi baseado na especificação *Standard Generalized Markup Language* (SGML, Linguagem de Marcação Generalizada Padrão) que é um método

---

[5] Programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. Mais em: <https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado>. Acesso em jan. 2016.

[6] Disponível em: <http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf>. Acesso em jan. 2016.

[7] É um consórcio internacional no qual organizações filiadas, uma equipe em tempo integral e o público trabalham juntos para desenvolver padrões para a *web*. Mais em: <http://www.w3c.br/Sobre>. Acesso em jan. 2016.

[8] Qualquer coisa que se coloca em uma página da *web* e que, quando clicada com o lado esquerdo do mouse, abre uma página diferente, ou um lugar diferente, da internet.

internacionalmente reconhecido e aceito contendo normas gerais para a criação de linguagens de marcação. Surge então a primeira versão do HTML.

Por basear-se no SGML, o HTML descreve a estrutura de um documento e não sua apresentação. A ideia parte do princípio que grande parte dos documentos possui elementos em comum, como títulos, parágrafos e listas, podendo desta forma identificar este conjunto de elementos e atribuir-lhes nomes apropriados, que se convencionou chamar de *tag* (em português, marca). O HTML define uma gama de elementos comuns às páginas *web*, tais como cabeçalhos, parágrafos, listas, tabelas, estilos de caracteres itálico e negrito, entre outros. Ao codificar uma página *web* no HTML as *tags* etiquetam os diferentes elementos da página dizendo "este é um cabeçalho" ou "esta é uma tabela" (LEMAY, 2002).

Com a evolução do HTML novas versões e variações surgiram, como exemplo o *eXtensible Hypertext Markup Language* (XHTML[9]). Cada versão lançada acrescentou novos recursos, *tags* e funcionalidades, possibilitando novas experiências de interação nos *web sites*. Desde a invenção da *web*, a HTML evoluiu por oito versões: HTML (1990), HTML+ (1993), HTML 2.0 (1994), HTML 3.0 (1995), HTML 3.2 (1997), HTML 4.0 (1997), HTML 4.01 (1999) e HTML 5 (2008) (FLATSCHART, 2011; SANDERS, 2012; SILVA, 2014a, b), sendo esta última a versão atual, ainda em fase de desenvolvimento.

De acordo com Silva (2014a), em maio de 2007 o W3C reconsiderou a decisão de encerrar o desenvolvimento da HTML em favor da XHTML e tornou pública sua decisão de retomar os estudos para o desenvolvimento da HTML5, tomando como base o trabalho que já vinha sendo desenvolvido pelo *Web Hypertext Application Technology Working Group* (WHATWG, Grupo de Trabalho para Tecnologias de Hipertexto em Aplicações para Web), que estavam insatisfeitos com o caminho que a *web* tomava com o rumo dado ao XHTML e por isso deram início ao desenvolvimento do que é hoje conhecido por HTML5 (FERREIRA; EIS, 2013; SILVA, 2014a).

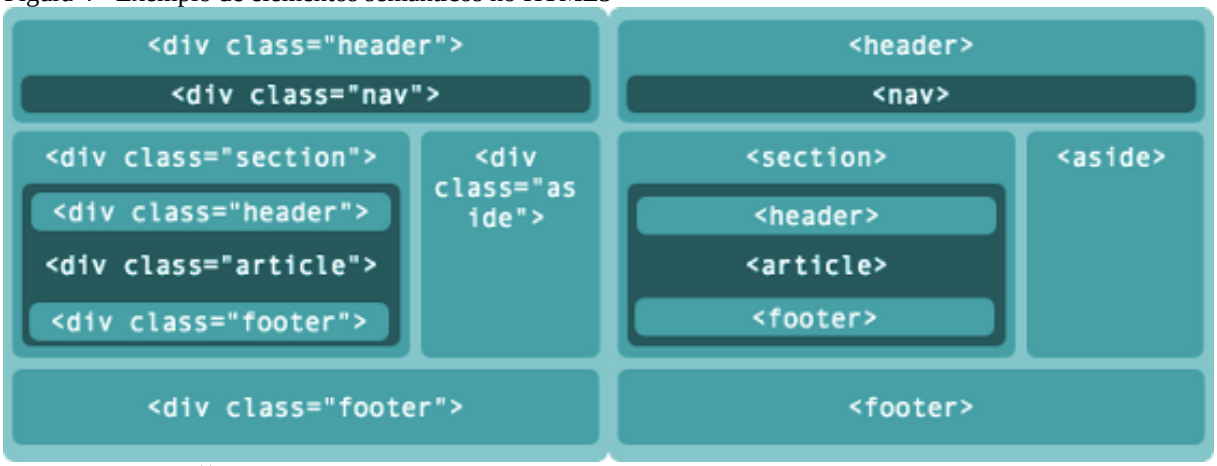
Dentre as mudanças presentes na nova versão estão a manipulação facilitada dos elementos que possibilita ao desenvolvedor modificar as características dos objetos de forma discreta e transparente para o usuário final, e um conjunto de ferramentas que permitem o CSS (seção 2.4.2) e JavaScript trabalharem da melhor maneira possível. Por meio de suas

[9] É uma família de atuais e futuros tipos de documentos e módulos que reproduzem, englobam e ampliam o HTML4. São baseados em XML e, por fim, são projetados para funcionar em conjunto com os agentes do usuário baseados em XML (W3C, 2000).

Application Programming Interfaces (APIs)[10] é possível manipular as características destes elementos, de modo que o *site* ou aplicação permaneçam leves e funcionais. O HTML5 também cria novas *tags* e modifica a função de outras. Versões anteriores não possuíam padrão para a criação de seções comuns e específicas como rodapé, cabeçalho, barra lateral, menus etc. Não existia um padrão de nomenclatura de *ids*, classes ou *tags*, nem um método capaz de capturar automaticamente informações localizadas nos rodapés dos *websites* (FERREIRA; EIS, 2013). O HTML5 incorporou novos elementos e atributos. A listagem completa pode ser conferida em http://www.w3schools.com/html/html5_new_elements.asp.

Outra característica importante no HTML5 é a semântica, que, segundo Silva (2014b, p. 50), significa "o uso do elemento apropriado ao conteúdo que marca". Tal característica modifica a maneira como se codifica e organiza a informação na página, como explica. Com mais semântica e menos código, provê mais interatividade sem a necessidade de instalação de *plug-ins* e perda de performance. Permite criação de código interoperável entre dispositivos que facilita o reuso da informação de diversas formas (FERREIRA; EIS, 2013, p. 10). Tudo isso implica em um código mais enxuto se comparado as versões do HTML4 ou XHTML, como destaca Silva (2014b, p. 50). A Figura 4 mostra um exemplo da semântica aplicada pelo HTML5. Observam-se elementos sem semântica e com semântica, respectivamente, nos códigos da esquerda e direita.

Figura 4 - Exemplo de elementos semânticos no HTML5



Fonte: UX.BLOG[11]

Por fim, embora o HTML5 ainda não esteja finalizado, o conteúdo disponível tem sido utilizado massivamente, melhorando a experiência tanto para desenvolvedores quanto para

10 Conjunto de padrões de programação que permitem a construção de aplicativos e a sua utilização. Mais em: <https://pt.wikipedia.org/wiki/Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es>. Acesso em jan. 2016.

11 Disponível em: <http://www.uxdesign.blog.br/front-end/html5-estrutura-semantica/>. Acesso em: jan. 2016.

usuários finais, tornando, portanto, mais fácil e intuitivo o desenvolvimento *web* (CAELUM, 2015, p. 36). O HTML5 veio para revolucionar o cenário da *World Wide Web*, trazendo mudanças na forma como se desenvolve conteúdos para a *web*.

2.4.2 CSS 3

CSS é a abreviação de *Cascading Style Sheet* (em português, Folhas de Estilo em Cascata) e é definido pela W3C (2016) como "um mecanismo simples para adicionar estilos aos documentos *web*." A informação entregue pelo HTML é formatada pelo CSS, que pode ser vários elementos diferentes tais como imagem, texto, vídeo, áudio ou qualquer outro elemento criado (W3C BRASIL, [2012?]). Em virtude das diferentes formatações aplicadas por cada navegador na interpretação do HTML surgiu a necessidade de criar um padrão capaz de fornecer informações aos navegadores sobre como os elementos do código seriam apresentados, problema este resolvido pelo surgimento do CSS.

Håkon Wium Lie e Bert Bos iniciaram o desenvolvimento do CSS em 1994, sendo lançada a recomendação oficial pelo W3C do CSS Level 1 (CSS1) em 1996. Em 2008 ocorre o lançamento do CSS2 (EIS, 2006). Antes do ano 2000 a W3C deu início ao desenvolvimento do CSS3 que atualmente ainda se encontra em criação e inclui facilidades como: selecionar primeiro e último elemento; selecionar elementos pares ou ímpares; selecionar elementos específicos de um determinado grupo de elementos; aplicar gradiente em textos e elementos; exibir bordas arredondadas; aplicar sombras em texto e elementos; permitir manipulação de opacidade; realizar controle de rotação; realizar controle de perspectiva; suportar animação; permitir estruturação independente da posição no código HTML; permitir duas imagens como *background* de um mesmo objeto; suportar fontes personalizadas; suportar múltiplas colunas de texto; dentre outras (W3C BRASIL, [2012?], p. 6).

A sintaxe CSS é simples. Seja o *seletor* o elemento/*tag* que se deseja estilizar, a *propriedade* uma característica do elemento a ser estilizada, e o *valor* a quantificação ou qualificação desta propriedade, a Figura 5 ilustra como é possível aplicar um estilo a um elemento usando CSS.

Figura 5 - Sintaxe do CSS

Fonte: (SILVA, 2012, p. 26)

Uma regra CSS pode conter várias declarações separadas por ponto e vírgula. O conjunto de regras é denominado *folha de estilo* e podem ser escritas segundo três métodos. O primeiro, denominado ***in-line***, permite escrever a folha de estilo na própria *tag* do elemento por meio do atributo *style* (ver QUADRO 2, linha 1). O segundo método é o **interno**, onde a folha de estilo é escrita no próprio documento HTML por meio da marcação <*style*> (ver QUADRO 2, linha 2). Por fim o terceiro método é o **externo**, onde um arquivo externo ao código HTML com extensão *.css* é referenciado no código por meio da marcação <*link*> (ver QUADRO 2, linha 3). O Quadro 2 apresenta a sintaxe dos três métodos em um trecho de código hipotético.

Quadro 2 - Métodos para aplicação do CSS

Método	**Sintaxe**
In-line	<body style=”background-color: #FFF;”>
Interno	<style type=”text/css”> aqui vão as regras CSS </style>
Externo	<link rel="stylesheet" type="text/css" href="css/estilo.css"/>

Fonte: Elaborado pelo autor

Os seletores podem ainda ser agrupados ou encadeados. No caso de encadeamento, os elementos são separados por espaço, e no agrupamento, os elementos são separados por vírgula. O Quadro 3, linha 1, ilustra um pequeno trecho de código encadeado, enquanto que o Quadro 3, linha 2 ilustra um exemplo com agrupamento de seletores. Uma lista atualizada com todos os seletores disponíveis até o CSS 3 e seus respectivos significados encontra-se disponível em https://www.w3.org/TR/css3-selectors/#selectors.

Quadro 3 - Exemplos de seletores encadeados e agrupados

Seletor	**Exemplo**

Encadeado	div p strong a {
	color: red;
	}

Agrupado	strong, em, span {
	color: red;
	}

Fonte: Adaptado de (W3C BRASIL, [2012?], p. 7-8)

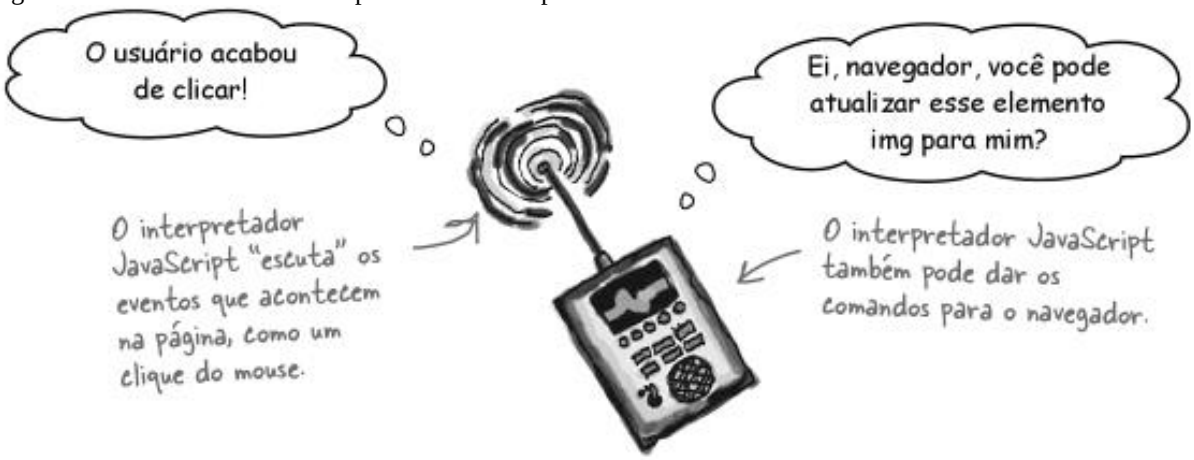
O principal benefício do CSS é a separação explícita entre o formato e o conteúdo do documento, ficando o CSS responsável somente pela função de apresentação, originando a consagrada frase que resume CSS + HTML (SILVA, 2012, p. 25): “HTML para estruturar e CSS para apresentar”. Porém, existem muitas outras vantagens no uso de CSS, das quais pode-se citar: economiza-se tempo de criação e manutenção pois isola os códigos de formatação aplicado a várias páginas HTML em um único arquivo *.css* externo; reduz código de descritores HTML da página; navegadores carregam conteúdo mais rapidamente; maior eficiência no gerenciamento do *layout*; rápido aprendizado para aprender a escrever apenas estilos; *design* sofisticado sem utilização de imagens e tabelas; e aumento da portabilidade dos documentos *web*. Por outro lado, a principal desvantagem deste padrão é o fato de que nem todos os navegadores suportam todas as especificações definidas pelo W3C. Além disso cada navegador implementa os estilos de forma não padronizada, resultando em documentos apresentados de maneiras diferentes em cada navegador (CCUEC, 2002).

Contudo, o CSS3 é peça fundamental para o futuro da *World Wide Web*. Seus diversos recursos e funcionalidades mudaram a forma de se apresentar *sites* e aplicações na internet, exibindo páginas cada vez mais elegantes e agradáveis ao usuário. Com os inúmeros avanços e o suporte prestado pelos navegadores aumentando, o CSS tornou-se a arma mais poderosa para os *designers web* (W3C BRASIL, [2012?], p. 6).

2.4.3 jQuery

Segundo Chaffer e Swedberg (2007), a *World Wide Web* é um ambiente dinâmico, que permite obter um padrão elevado para estilo e funcionalidades presentes nos *sites* (como animação, interação, dentre outros) desejado pelos seus usuários. HTML e CSS fornecem estrutura e estilo às páginas *web*, porém são estáticas após renderização. Para alterar páginas dinamicamente sem recarregar seu conteúdo é necessário que haja comunicação com o navegador através de uma linguagem de *script*. A linguagem mais popular deste tipo é JavaScript. Praticamente todo navegador possui o interpretador JavaScript embutido, que interpreta as instruções do *script* e as converte em ações dinâmicas na página (BENEDETTI; CRANLEY, 2013). O funcionamento de um interpretador JavaScript é descrito brevemente na Figura 6.

Figura 6 - Funcionamento do interpretador JavaScript



Fonte: (BENEDETTI; CRANLEY, 2013, p. 5)

Neste sentido, equipes de desenvolvedores se empenharam no desenvolvimento de bibliotecas JavaScript para automatizar tarefas comuns e simplificar as complexas, possibilitando a criação de *sites* mais interessantes e interativos (CHAFFER; SWEDBERG, 2007, p. 5, tradução do autor). jQuery é um exemplo dessas poderosas bibliotecas JavaScript que podem melhorar o desenvolvimento de *websites* independentemente do seu *background*[12]. Foi criada por John Resig e lançada no final de 2006. Trata-se de um projeto de código aberto mantido por uma equipe de desenvolvedores JavaScript que fornece uma ampla gama de recursos, sintaxe de fácil aprendizado e uma robusta compatibilidade entre plataformas. Inúmeros *plug-ins* foram e continuam sendo desenvolvidos, estendendo cada vez mais as funcionalidades do jQuery, tornando-a uma ferramenta essencial para manipulação de *scripts* do lado cliente (CHAFFER; SWEDBERG, 2007). O Quadro 4 apresenta um exemplo simples de *script* jQuery.

Quadro 4 - Exemplo de script jQuery

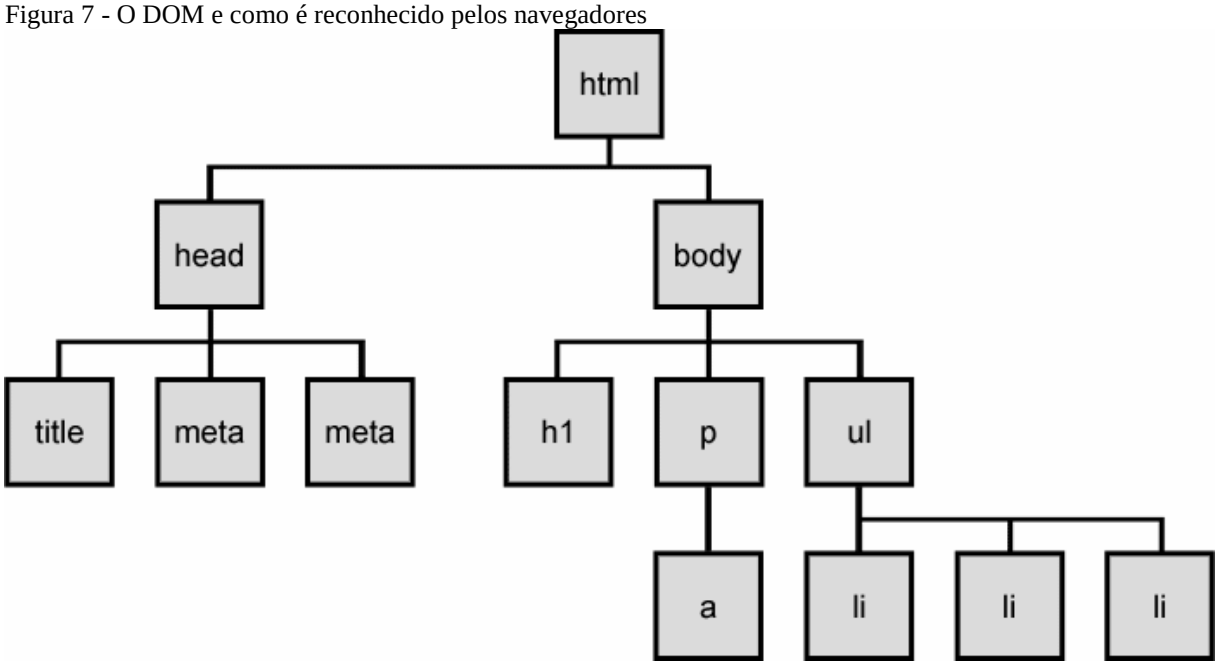
```
// Quando o documento da página estiver pronto
$(document).ready(function() {
        // Quando qualquer elemento button é clicado
        $("button").click(function() {
                // Exibe uma janela de alerta
                alert("Você clicou em um botão.");
        });
});
```

Fonte: Elaborado pelo autor
O sinal do cifrão com parênteses é a abreviação da função jQuery. Ao invés de escrever jQuery() escreve-se $()

[12] O que constitui o pano de fundo, isto é, o que está em segundo plano, como fotos, vídeos, imagens etc.

Contudo, para entender como o jQuery consegue alterar as páginas *web*, é preciso observar o funcionamento do processo de exibição das mesmas pelo navegador. Todo navegador usa o *Document Object Model* (DOM, Modelo de Objetos de Documento) para construir uma página a partir do HTML simples e o código CSS em uma página clicável completa com texto, imagens, vídeos etc. O DOM representa como as marcações HTML, XHTML e XML são organizadas e lidas pelo navegador, provendo um esquema padronizado onde todos os navegadores o usam para tornar a navegação *web* mais eficaz (BENEDETTI; CRANLEY, 2013). As marcações, uma vez indexadas pelo DOM, são transformadas em elementos, os quais são dispostos numa estrutura de árvore, conforme mostra a Figura 7.

Figura 7 - O DOM e como é reconhecido pelos navegadores

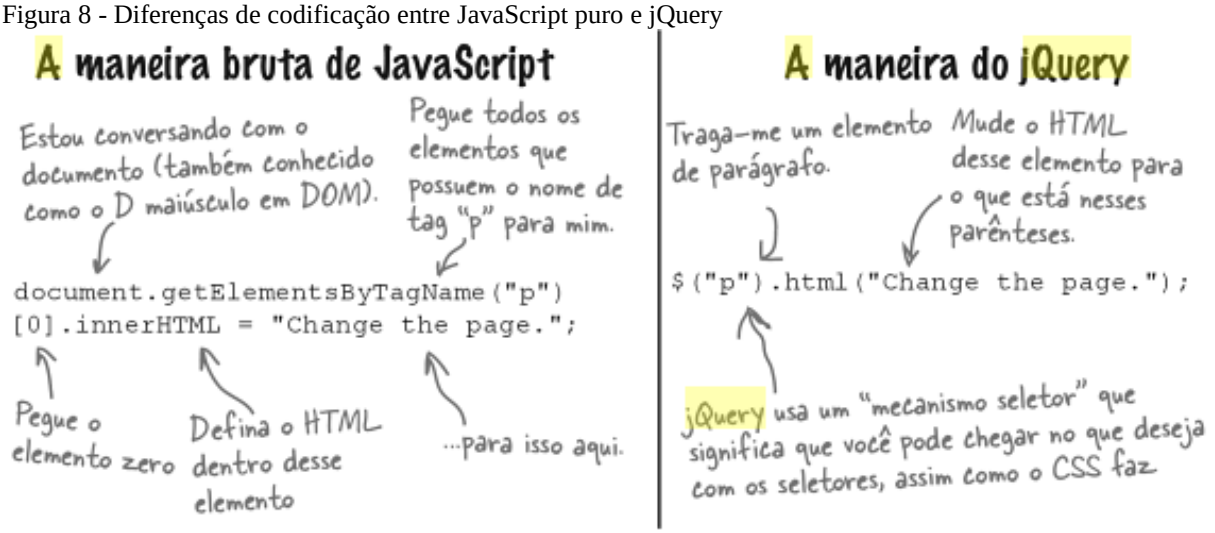


Fonte: TABLELESS[13]

É neste ponto que o jQuery facilita a vida dos desenvolvedores. Embora o DOM possa parecer complexo, a biblioteca jQuery o mantém simples, permitindo que o desenvolvedor trabalhe com o DOM sem saber detalhadamente tudo por trás do mesmo, deixando que por baixo de tudo isso o JavaScript faça o trabalho pesado (BENEDETTI; CRANLEY, 2013). A Figura 8 mostra um exemplo simples da diferença de codificação entre JavaScript puro e jQuery.

[13] Disponível em: http://tableless.com.br/tenha-o-dom/. Acesso em jan. 2016.

Figura 8 - Diferenças de codificação entre JavaScript puro e jQuery



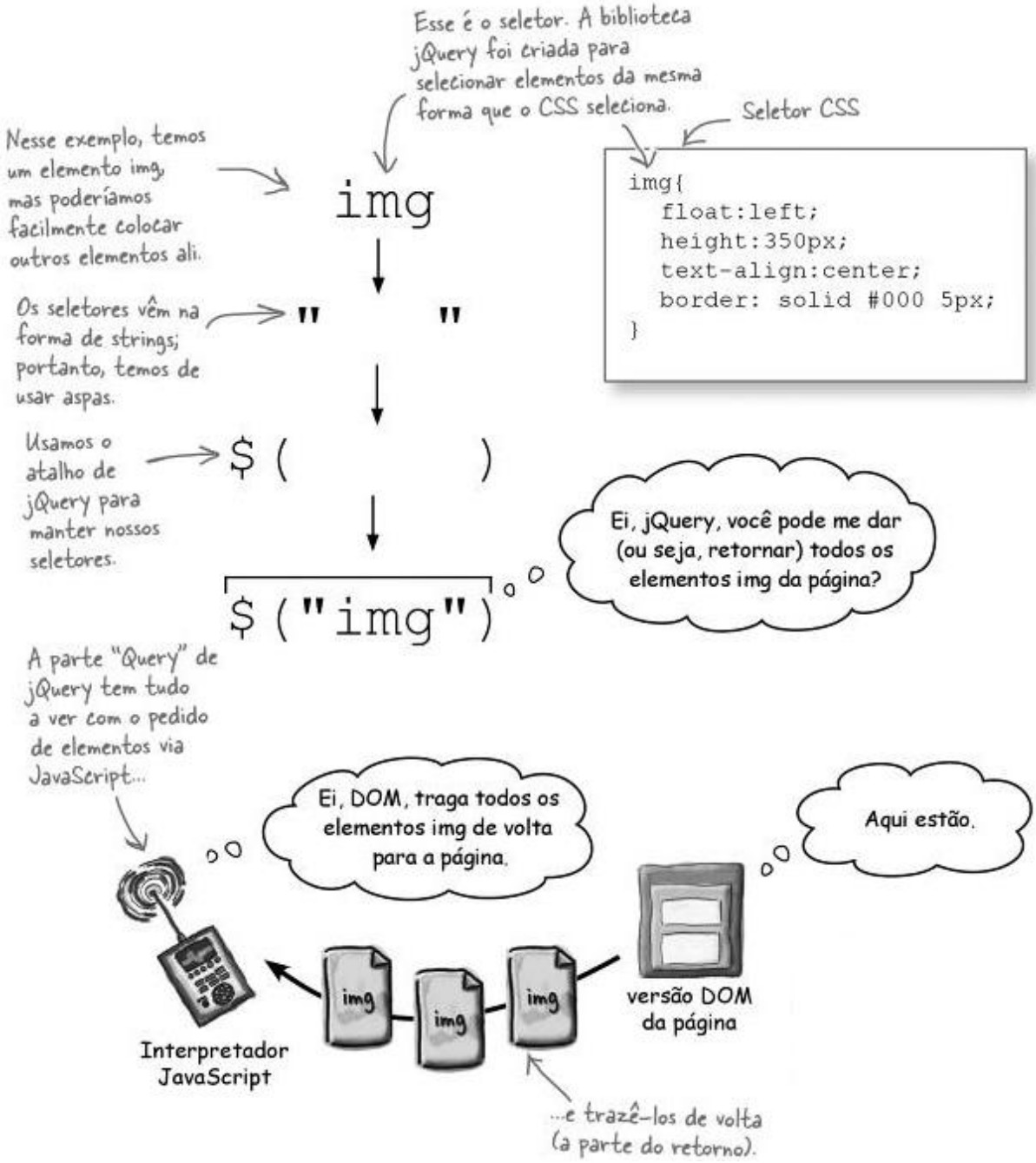
Fonte: (BENEDETTI; CRANLEY, 2013, p. 9)

Para alterar, por exemplo, cinco elementos de parágrafo (*tag* <p>) na página via JavaScript puro é necessário implementar a iteração por um laço de repetição para mudar os elementos; em jQuery, a implementação seria tal como ao apresentado na Figura 8. Uma estratégia seguida pela biblioteca consiste em trabalhar com conjuntos de objetos ao invés de objetos individuais. Segundo Chaffer e Swedberg (2007), esta técnica é denominada de *iteração implícita*, que reduz as construções de *loops*, diminuindo consideravelmente o tamanho do código.

A alma por trás do jQuery está no uso de **seletores**. Para utilizar o jQuery com toda sua robustez e eficiência, é necessário saber utilizá-los corretamente. A grande vantagem é que a biblioteca jQuery utiliza os mesmos seletores CSS, ou seja, pode-se recuperar um conjunto de elementos por meio de suas *tags* ou propriedades como classe, id etc. A diferença aqui é que seletores CSS selecionam elementos para estilizá-los, enquanto que seletores jQuery selecionam elementos para atribuir-lhes comportamento. Com isso, o nome jQuery tem tudo a ver com *query* (consulta), pois pede-se algo por meio de um seletor e o interpretador JavaScript pede ao DOM para buscá-lo (BENEDETTI; CRANLEY, 2013).

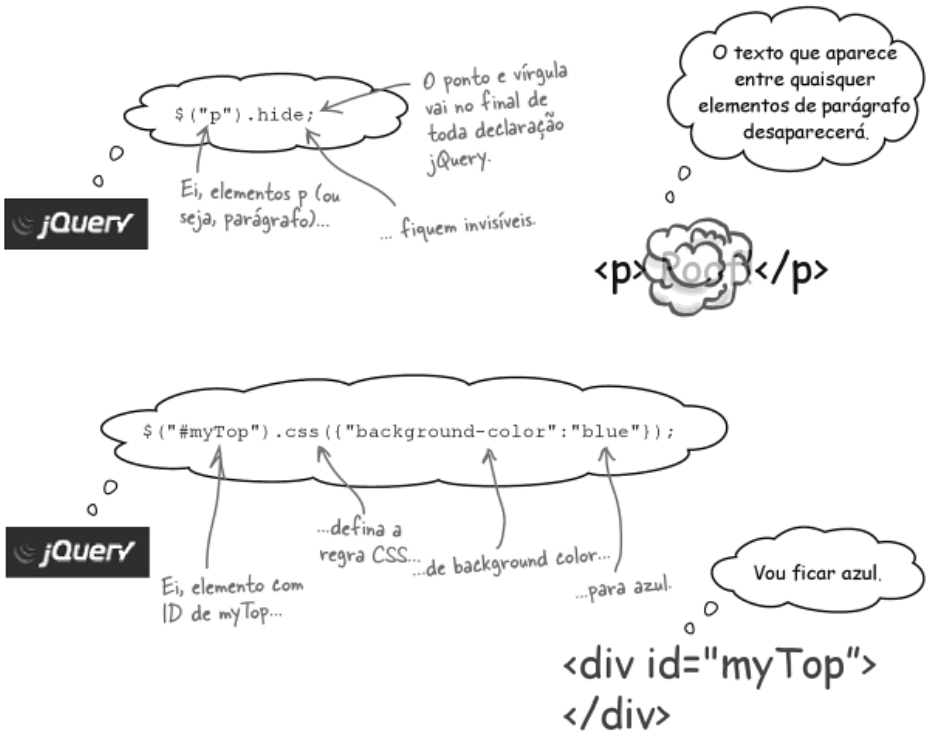
Para melhor compreender o entendimento sobre como o jQuery opera em conjunto com o DOM, as Figuras 9 e 10 ilustram graficamente o processo de recuperação de elementos, assim como elementos estáticos na página podem ser alterados dinamicamente pelo jQuery, respectivamente.

Figura 9 - Processo de recuperação de elementos realizado pelo jQuery



Fonte: (BENEDETTI; CRANLEY, 2013, p. 15)

Figura 10 - Alteração de elementos estáticos pelo jQuery



Fonte: (BENEDETTI; CRANLEY, 2013, p. 16)

Desta forma, a biblioteca jQuery oferece diversos recursos, os quais Chaffer e Swedberg (2007, p. 6, tradução do autor) destacam como principais:

- **Acessar partes da página**: sem uma biblioteca JavaScript, muitas linhas de código são escritas para percorrer a árvore DOM e localizar partes específicas da estrutura de um documento HTML. Visto isso, jQuery oferece um mecanismo de seleção robusto e eficiente para retornar exatamente o pedaço do documento que será inspecionado ou manipulado.
- **Modificar a aparência da página**: CSS oferece um poderoso método de influenciar o modo como o documento HTML é renderizado, porém fica aquém quando os navegadores não suportam todos os mesmos padrões. jQuery pode solucionar este problema, fornecendo os mesmos padrões de apoio em todos os navegadores. Além disso, pode mudar as classes ou propriedades de estilos individuais aplicadas a uma parte do documento mesmo após esta ter sido renderizada.
- **Alterar o conteúdo da página**: jQuery pode modificar o conteúdo de um documento com algumas teclas. Texto pode ser alterado, imagens podem ser inseridas ou trocadas, listas podem ser reordenadas, ou toda a estrutura do HTML pode ser reescrita e estendida, tudo com uma única API fácil de usar.

- **Responder à interação do usuário com uma página**: até mesmo os comportamentos mais elaborados e poderosos não são úteis se não é possível controlar quando eles ocorrem. A biblioteca jQuery oferece uma maneira elegante para interceptar uma grande variedade de eventos, como por exemplo o clique em um *link*, sem a necessidade de confundir o código HTML com manipuladores de eventos.
- **Adicionar animação à página**: a fim de implementar efetivamente comportamentos interativos, o projetista deve também prover um *feedback* visual para o usuário. A biblioteca jQuery facilita isto fornecendo uma variedade de efeitos, tais como *fades*[14] e *wipes*[15], bem como um kit de ferramentas para a elaboração de novos.
- **Recuperar informação do servidor sem atualizar a página**: Este padrão de código, cunhado por *Asynchronous JavaScript and XML* (AJAX) (para mais detalhes sobre AJAX, ver seção 2.4.4), auxilia desenvolvedores *web* na elaboração de um *site* responsivo e rico em recursos. jQuery remove a complexidade específica do navegador a partir deste processo, permitindo que desenvolvedores concentrem-se nas funcionalidades do lado servidor.
- **Simplificar tarefas JavaScript comuns**: em adição a todas as características específicas do documento do jQuery, a biblioteca provê melhorias para construções básicas de JavaScript, como iteração e manipulação de vetores.

Para utilizar o jQuery, basta fazer o *download* do arquivo JavaScript no *site*[16] mantenedor e adicioná-lo no documento HTML por meio da *tag* <script>. O arquivo está disponível em duas versões: *Compressed* e *Uncompressed*. A primeira se trata de uma versão do código compactada em uma única linha, e deve ser utilizada em produção, para não sobrecarregar a transferência de dados entre o servidor e os navegadores. Durante o desenvolvimento, pode-se utilizar a segunda, pois esta possui um código descompactado e bem mais legível, facilitando a edição e depuração.

Se comparado a outras bibliotecas JavaScript, jQuery visa mudar a maneira como os desenvolvedores *web* pensam sobre criar ricas funcionalidades em suas páginas. Ao invés de perder tempo com a complexidade do JavaScript puro, projetistas podem aprimorar seus

[14] Este efeito pode ser entendido como uma transição suave entre duas imagens com a opacidade de uma diminuindo até se tornar invisível e revelando a de baixo.
[15] Efeito de mudança gradual de uma imagem para outra. Ex: margem simples, um círculo estendido etc.
[16] Disponível em: <https://jquery.com/download/>. Acesso em jan. 2016.

conhecimentos existentes em CSS, HTML, XHTML e o bom e velho simples JavaScript para manipular os elementos da página diretamente, tornando o desenvolvimento mais rápido uma realidade (BIBEAULT; KATZ, 2008).

A biblioteca jQuery está presente em diversos *frameworks* para desenvolvimento *front-end*, como exemplo o *Bootstrap* (discutido na seção 2.4.5), que juntando os recursos supracitados, com aqueles específicos de cada *plug-in* desenvolvido, facilitam veemente para o desenvolvimento de *websites* mais interativos, provendo uma experiência cada vez mais satisfatória para o usuário.

2.4.4 AJAX

Nos primórdios da *web* para atualizar o conteúdo de uma página era necessário que o navegador recarregasse todo o conteúdo (código HTML modificado, estilos e scripts) a partir do servidor, em uma operação lenta com intervenção do usuário. Para exemplificar, seja uma aplicação de *web mail*. Para saber se haviam novos e-mails na caixa de entrada o usuário era obrigado a realizar uma recarga manual, ineficiente, onde o servidor reconstruía todo o código HTML da página, CSS e JavaScript, reenviando o novo conteúdo ao navegador. Nesta situação o ideal seria o servidor enviar apenas as novas mensagens de e-mail, e não todo o conteúdo. Em 2003, os principais navegadores contornaram este problema com a adoção da API *XMLHttpRequest*[17] (XHR), que possibilitava os navegadores se comunicarem com o servidor sem a necessidade de recarregar a página (JQUERY, 2015). O objeto XHR é a parte principal da tecnologia conhecida como *Asynchronous JavaScript and XML* (AJAX, em português, Java e XML assíncrono). Trata-se de um objeto JavaScript que pode ser usado para fazer requisições ao servidor em segundo plano, evitando que o mesmo fique congelado ou que a página atual seja recarregada por inteiro.

Garret (2005) afirma que AJAX não é uma única tecnologia, e sim um conjunto de várias. Para o autor, AJAX incorpora padrões de apresentação utilizando XHTML e CSS; exibição dinâmica e interativa utilizando DOM; intercâmbio e manipulação de dados usando XML e *eXtensible Stylesheet Language for Transformation* (XSLT, linguagem extensível para

[17] Projetada originalmente pela Microsoft e adotada por outras empresas como Mozilla, Apple e Google. Atualmente está sendo padronizada pelo WHATWG. Mais em: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>. Acesso em jan. 2016.

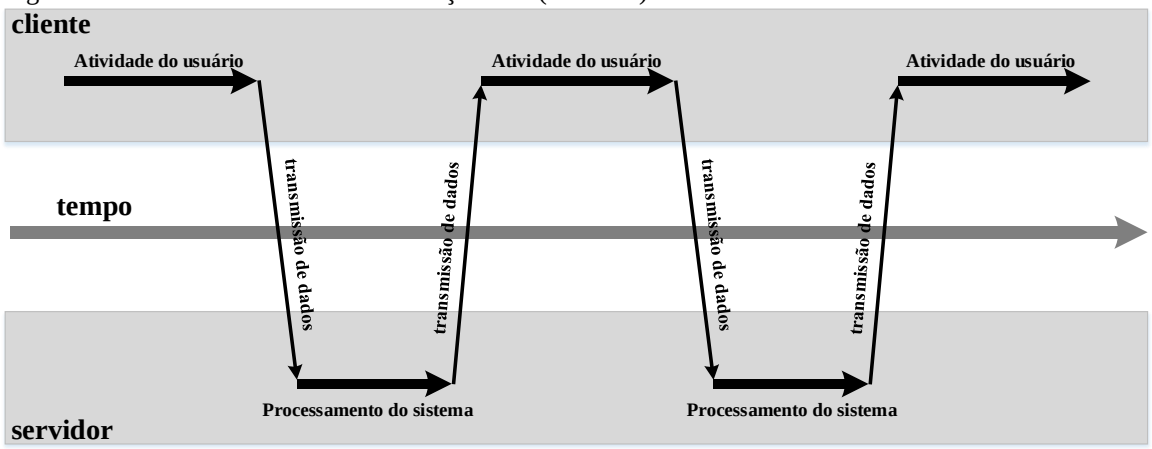
folhas de estilo de transformações); recuperação de dados assíncrona usando XMLHttpRequest; e JavaScript, que junta tudo isso.

A grande vantagem do AJAX está no fato de que ele é capaz de realizar essa comunicação de maneira assíncrona, ou seja, enquanto a solicitação é processada pelo servidor, todo o restante da página continua executando normalmente, de modo que a interação com o usuário não fica bloqueada. Em consequência dessa comunicação assíncrona realizada pelo AJAX, Limeira (2006, p. 16-17) destaca outras vantagens, a saber:

- **menor uso de banda**: menos informações irão trafegar na rede, diminuindo sensivelmente o uso de banda e tornando desta forma a navegação mais rápida.
- **respostas mais rápidas**: usuários não precisam esperar que toda a página seja processada a cada vez que ocorrer uma solicitação. O usuário pode interagir com uma parte da página e enquanto aguarda o retorno do servidor, pode interagir com as demais seções, tornando a navegação mais rápida e eficiente.
- **mais interatividade**: não há a necessidade de usar recursos pesados de programação ou *softwares* proprietários, é possível disponibilizar aplicações com alta capacidade de interatividade e usabilidade. As interfaces são amigáveis para o usuário final.

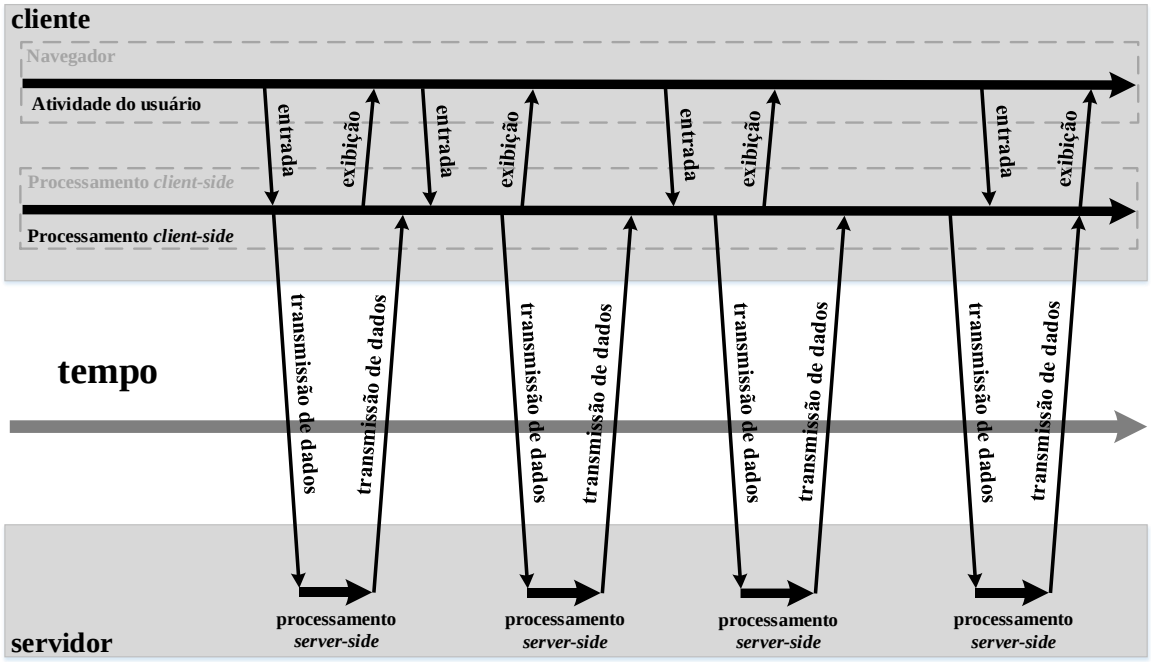
Sobre o modelo de comunicação síncrono na web, a Figura 11 ilustra graficamente a natureza *start-stop-start-stop* de interação na *web*, onde após realizar uma requisição, a atividade do usuário é interrompida até a resposta do servidor. Em contrapartida, a Figura 12 mostra o modelo de comunicação assíncrona na *web* proporcionada pelo AJAX. Neste modelo o usuário nunca está olhando para uma janela do navegador em branco e um ícone de ampulheta, esperando pelo servidor para fazer algo, sua atividade é contínua no tempo (GARRET, 2005).

Figura 11 - Modelo clássico de comunicação *web* (síncrono)



Fonte: Adaptado de (GARRET, 2005, p. 3)

Figura 12 - Modelo AJAX de comunicação *web* (assíncrono)



Fonte: Adaptado de (GARRET, 2005, p. 3)

Apesar das suas vantagens, os recursos do AJAX não devem ser utilizados sem um contexto adequado. Por exemplo, imagine se para renderizar uma página inteira, ao invés de realizar uma única requisição solicitando todo o código HTML, várias requisições AJAX sejam feitas pedindo o HTML de cada pedaço da página. O resultado não será eficiente, a aplicação terá mais código escrito e será mais pesada. É preciso ter bom senso na hora de aplicar o AJAX.

Para utilizar o AJAX, é necessário instanciar um objeto *XMLHttpRequest* (XHR). Ele é quem trocará informações com o servidor web. Essa troca normalmente acontece utilizando XML, no entanto, o objeto XHR permite o uso de qualquer formato de texto, como HTML, texto plano e o comumente utilizado JSON (*JavaScript Object Notation*). Os Quadros 5 e 6 mostram os principais métodos e propriedades que compõem este objeto.

Quadro 5 - Principais métodos do XHR (continua)

Métodos	**Descrição**
open(método, url, assinc, usuário, senha)	Esse método relaciona o objeto à página *web* que se deseja conectar. O argumento *método* pode ser *GET*, *POST* ou *PUT*. O endereço da url pode ser relativo ou absoluto. Os 3 últimos parâmetros são opcionais.

Quadro 5 - Principais métodos do XHR (conclusão)

Métodos	Descrição
send(content)	Envia a solicitação para o Servidor. Caso a conexão tenha sido aberta com o parâmetro *assinc* igual a *false* (indicando que a conexão não é assíncrona), esse método aguarda a resposta do servidor; caso contrário, não há espera (o que deve ser o padrão para aplicações AJAX).
setRequestHeader	Configura o cabeçalho http especificado com o valor fornecido.
getResponseHeader	Retorna o valor da *string* do cabeçalho especificado.
getAllResponseHeaders	Retorna uma *string* com todos os cabeçalhos http especificados.
abort	Interrompe o processamento atual do objeto XMLHttpRequest.

Fonte: (ASLESON; SCHUTTA, 2006, p. 25 *apud* LIMEIRA, 2006)

Quadro 6 - Principais propriedades do XHR

Propriedade	Descrição
status	Contém o código de status enviado pelo servidor *web*. (isto é, 200 para *OK*, 404 para Não Encontrado e assim por diante)
statusText	A versão em texto do código de status http. (isto é, *OK* ou *Not Found* e assim por diante)
readyState	O estado da solicitação. Os cinco valores possíveis são: 0 - não inicializada, 1 - carregando, 2 - carregada, 3 - interativa e 4 - concluída.
responseText	A resposta do servidor na forma de uma *string*.
responseXML	A resposta do servidor em formato XML.
onreadystatechange	O manipulador de eventos que é acionado a cada mudança de estado, normalmente, uma chamada a uma função JavaScript.

Fonte: (SOARES, 2006, p. 88-92 *apud* LIMEIRA, 2006)

O Quadro 7 mostra um trecho de código exemplo de comunicação usando AJAX.

Quadro 7 - Código exemplo de comunicação usando AJAX (continua)

```
// instancia o objeto XHR
var xhr = new XMLHttpRequest();

// função chamada quando o estado da solicitação feita pelo objeto XHR muda
xhr.onreadystatechange = function() {
    // verifica se a solicitação foi concluída sem erros
    if (xhr.readyState == 4 && xhr.status == 200) {
```

Quadro 7 - Código exemplo de comunicação usando AJAX (conclusão)

```
        // substitui o conteúdo da tag HTML de id “demo” com a resposta do objeto XHR
        document.getElementById("demo").innerHTML = xhr.responseText;
    }
};
// abre a conexão, relacionando o objeto XHR com à página em questão
xhr.open("GET", "ajax_info.txt", true);

// envia a solicitação para o servidor
xhr.send();
```

Fonte: Adaptado de W3Schools[18]

O trecho de código apresentando no Quadro 7 pode parecer complexo pois está escrito em JavaScript puro. A biblioteca jQuery torna o trabalho com AJAX muito mais simples e legível. Veja a versão do mesmo trecho de código escrita em jQuery apresentada pelo Quadro 8.

Quadro 8 - Versão do código AJAX em JavaScript escrito em jQuery

```
// função do jQuery para realizar uma requisição via AJAX
$.ajax({
    // indica a url fonte dos dados
    url: ‘ajax_info.txt’,
    // função executada se a solicitação foi concluída com êxito
    sucess: function(resposta) {
        $(‘#demo’).html(resposta);
    },
    // função executada se ocorreu algum erro durante a comunicação
    error: function(erro) {},
    // função sempre executada, independentemente de erro ou sucesso na comunicação
    complete: function() {}
});
```

Fonte: Elaborado pelo autor

A utilização do AJAX em conjunto com bibliotecas como jQuery torna-se mais fácil e produtiva, facilitando sua adaptação. Dentre as facilidades para os desenvolvedores, estão: atualizar a página sem recarregá-la, solicitar dados ao servidor após a página ter sido carregada, receber dados do servidor após a página ter sido carregada, enviar dados para o servidor em segundo plano. Trata-se de uma tecnologia fortemente recomendada na maioria das aplicações e *sites web*. Porém, deve-se ter cautela no seu uso, cabendo aos desenvolvedores perceberem o melhor momento para aplicá-lo procurando assim aproveitar o máximo de todos os benefícios que o AJAX pode proporcionar.

[18] Disponível em: <http://www.w3schools.com/ajax/>. Acesso em: jan. 2016.

2.4.5 Bootstrap

Segundo Silva (2015, p. 20),

> *Bootstrap* é o mais popular *framework* JavaScript, HTML e CSS para desenvolvimento de *sites* e aplicações *web* responsivas e alinhadas com a filosofia *mobile first*. Torna o desenvolvimento *front-end* muito mais rápido e fácil. Indicado para desenvolvedores de todos os níveis de conhecimento, dispositivos de todos os tipos e projetos de todos os tamanhos.

Em agosto de 2011, Mark Otto, desenvolvedor trabalhando no Twitter[19], juntamente com Jacob Thornton anunciaram ao mundo o lançamento do *framework Bootstrap*. O que motivou sua criação foram as inconsistências geradas por diferentes bibliotecas utilizadas por cada desenvolvedor do Twitter para desenvolvimento de *front-end*, o que dificultava a integração, escalabilidade e manutenção das aplicações criadas pelos desenvolvedores da equipe. Sua primeira versão estável foi apresentada na semana de 22 a 29 de Outubro de 2011 na primeira Twitter Hackweek (SILVA, 2015).

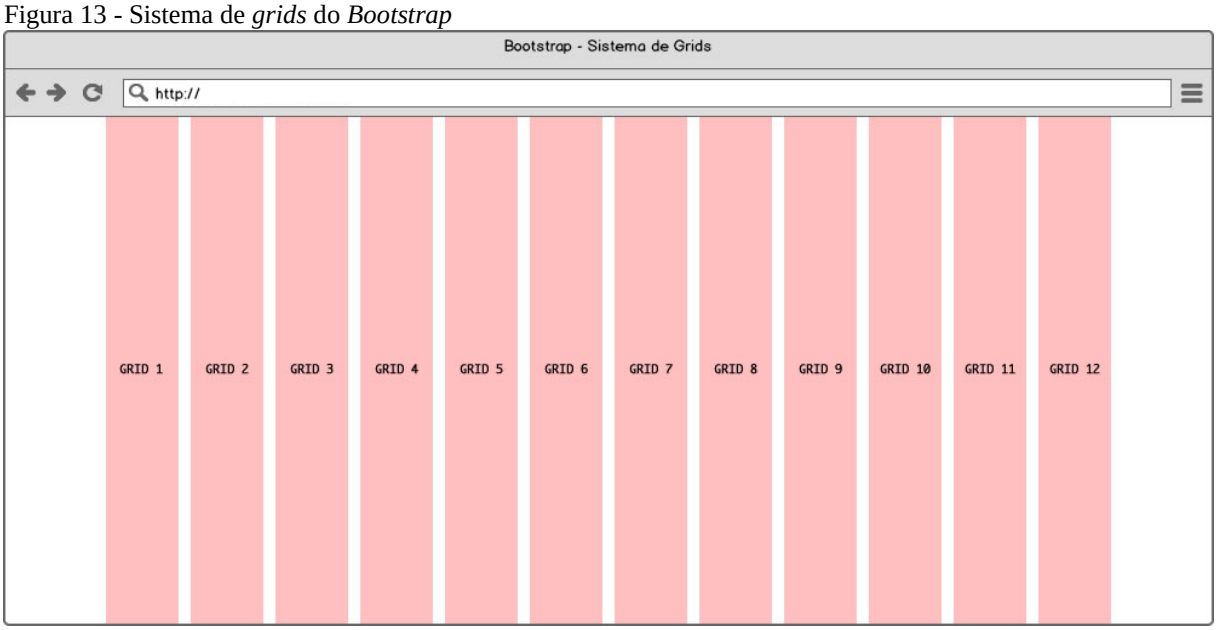
Os estilos providos pelo *framework* são a sua essência. O CSS é criado com LESS[20] e *Syntactically Awesome StyleSheets* (SASS)[21], que são pré-processadores destinados a gerar folhas de estilos CSS capazes de oferecer mais flexibilidade e poder que as tradicionais não processadas, como explica Silva (2015). O autor ainda comenta que LESS e SASS fornecem uma vasta gama de funcionalidades, como declarações CSS aninhadas, variáveis para valores de propriedades CSS, *mixins* (tipo de classe que permite reuso), operadores e funções de declarações de cores. Como qualquer outra ferramenta, o *Bootstrap* também possui suas vantagens e desvantagens. Nascimento (2013) destaca algumas vantagens: possui documentação detalhada e de fácil entendimento; é otimizado para o desenvolvimento de *layouts* responsivos; possui componentes suficientes para o desenvolvimento de qualquer *site* ou sistema *web* com interface simples; facilita a criação e edição de *layouts* por manter padrões; funciona em todos os navegadores atuais (Chrome, Safari, Firefox, IE, Opera, Edge). Como desvantagens o autor aponta: o código deverá seguir os "padrões de desenvolvimento *Bootstrap*"; tema padrão e comum do *Bootstrap*, sem ajustes visuais, que faz com que o projeto se pareça com outros que também utilizam o *Bootstrap*.

---

[19] Disponível em: <https://twitter.com/>. Acesso em jan. 2016.
[20] Disponível em: <http://lesscss.org/>. Acesso em jan. 2016.
[21] Disponível em: <http://sass-lang.com/>. Acesso em jan. 2016.

O *Bootstrap* é compatível com HTML5 e CSS3 e possui um sistema de *grids*, o qual permite a divisão do conteúdo da página em até 12 colunas, como mostrado na Figura 13.

Figura 13 - Sistema de *grids* do *Bootstrap*



Fonte: Adaptado de (NASCIMENTO, 2013)

O sistema de *grids* do *Bootstrap* permite qualquer divisão, desde que o resultado da soma das larguras seja 12. Com isso, é possível mudar o visual do *site*/aplicação alterando apenas o valor da largura das colunas. E a grande vantagem deste sistema de *grids*, é que ele é nativamente responsivo, ou seja, o conteúdo se comporta de maneira diferente para tamanhos de resolução diferentes, exibindo adequadamente a interface em variados tipos de dispositivos (como *notebooks*, *tablets* e *smartphones*). Este comportamento é definido simplesmente com a adição de classes predefinidas nas *tags* HTML. O Quadro 9 lista estas classes, destacando os tipos de dispositivos adequados para usá-las e exemplos destes.

Quadro 9 - Classes do *Bootstrap* para comportamento responsivo

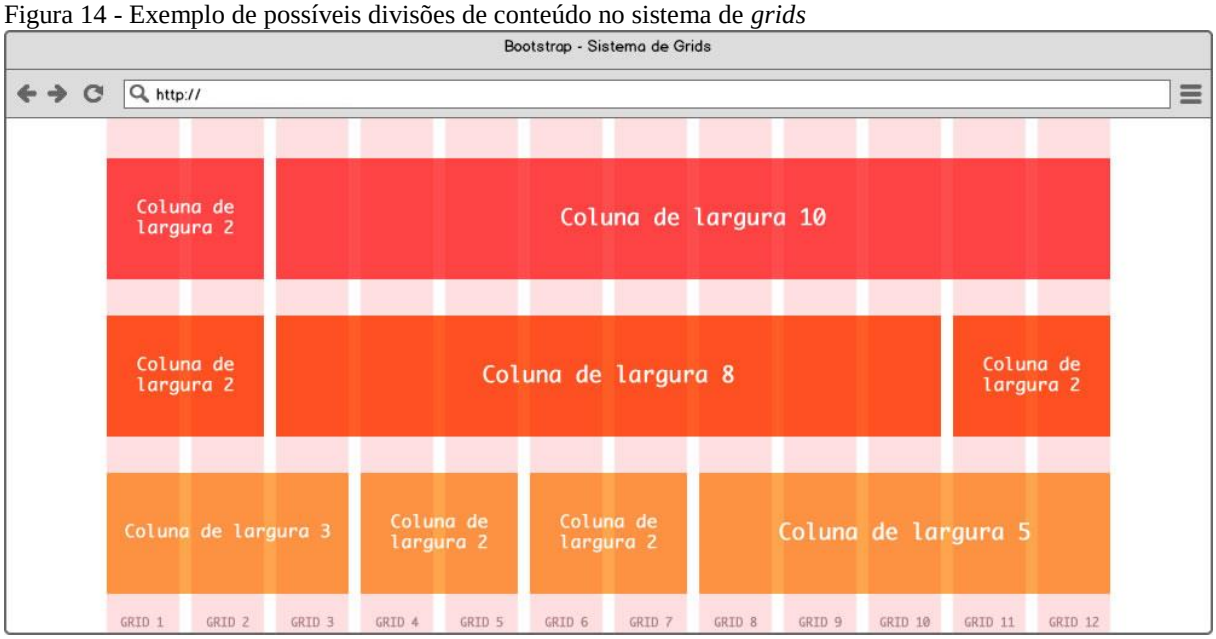
Classe	**Tipo de dispositivo**	**Exemplo**
.col-xs-*[1]	extra pequenos, com resolução menor que 768px	celulares
.col-sm-*	pequenos, com resolução maior ou igual a 768px	*tablets*
.col-md-*	médios, com resolução maior ou igual a 992px	*desktops* e *notebooks*
.col-lg-*	grandes, com resolução maior ou igual a 1200px	*desktops* e *notebooks*

Fonte: Adaptado de (SILVA, 2015, p. 57)

A Figura 14 mostra um exemplo de configuração de *layout* possível usando o sistema de *grids*, com conteúdos dispostos em diferentes divisões. Na Figura 15 é possível observar como o *layout* de uma página se adequa ao tamanho reduzido de telas, alterando a disposição

de elementos como menus e barras laterais, ocultando-os quando em dispositivos ainda menores, como no caso de *smarthphones*.

Figura 14 - Exemplo de possíveis divisões de conteúdo no sistema de *grids*



Fonte: Adaptado de (NASCIMENTO, 2013)

Figura 15 - Comportamento em diferentes resoluções

Fonte: (NASCIMENTO, 2013)

O *Bootstrap* contém ainda diversos componentes e *plug-ins* JavaScript. Exemplos de componentes presentes são: menus *dropdown*, botões, agrupamento de botões, barra de navegação, paginação, *breadcrumbs*[22], alertas, entre muitos outros. Caso o projeto faça uso dos *plug-ins* nativos do *framework*, é necessário a integração com a biblioteca jQuery. A lista completa de componentes e *plug-ins* disponibilizados pelo *Bootstrap* pode ser conferida no próprio *site*[23] do *framework*.

[22] Sistema de navegação estrutural usado para fornecer ao usuário um meio de localização dentro da estrutura de navegação do *site*. Mais em: <http://www.globalad.com.br/blog/o-que-sao-breadcrumbs/>. Acesso em jan. 2016.
[23] Disponível em: <http://getbootstrap.com/javascript/>. Acesso em jan. 2016.

Para utilizar o *Bootstrap* no desenvolvimento do *site*/aplicação *web*, basta fazer o *download*[24] do *framework* e integralizá-lo ao projeto. No entanto, o *site* do *Bootstrap* disponibiliza três versões para *download*: *Bootstrap*, *Source* e *SASS*. A versão *Bootstrap* é a menor requerida para usufruto de todas as funcionalidades, conta com CSS pré-compilado e minificado, além dos *scripts* JavaScript e fontes; contudo, nenhuma documentação ou arquivos fontes originais estão incluídos. A versão *Source* contém além do CSS pré-compilado, JavaScript e fontes, toda a documentação e arquivos fontes. A versão *SASS* é destinada para uso com o pré-processador SASS (SILVA, 2015). Para adicioná-lo ao projeto, basta incluir a folha de estilo CSS ao documento HTML. Caso haja necessidade de utilizar os *plug-ins* JavaScript nativos é necessário adicionar a biblioteca jQuery ao projeto, pois os *scripts* nativos são desenvolvidos em jQuery. O Quadro 10 apresenta um *template* básico para uma aplicação que faz uso do *Bootstrap*.

Quadro 10 - Exemplo de *template* básico para utilização do *Bootstrap* (continua)

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
         <!-- Estas 3 meta tags devem obrigatoriamente ser as primeiras na seção head -->
          <!-- As demais devem vir depois delas -->

        <title>Template básico Bootstrap</title>

        <!-- CSS do Bootstrap -->
        <link href="css/bootstrap.min.css" rel="stylesheet">

        <!-- HTML5 shim e Respond.js para suporte dos elementos HTML5 e das media
        queries ao IE8 -->
<!-- OBS: Respond.js não funciona em páginas carregadas com uso do protocolo file:// -->
        <!--[if lt IE 9]>
            <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
            <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
          <![endif]-->
    </head>
    <body>
        <h1>Hello, world!</h1>
        <!-- jQuery (caso se use plugins JavaScript do Bootstrap) -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
        </script>
```

[24] Disponível em: <http://getbootstrap.com/getting-started/#download>. Acesso em jan. 2016.

Quadro 10 - Exemplo de *template* básico para utilização do *Bootstrap* (conclusão)

```
        <!-- Inclui todos os plug-ins compilados -->
        <script src="js/bootstrap.min.js"></script>
    </body>
</html>
```

Fonte: Adaptado do *Site* do *Bootstrap*[25]

Por fim, é notável o aumento de produtividade no desenvolvimento *web* com o uso do *Bootstrap*, em especial pelas facilidades que ele proporciona na criação e edição de *layouts* responsivos. Devido à simplicidade, é recomendado para desenvolvedores iniciantes e avançados. Com a variedade de recursos e componentes presentes neste *framework* é possível desenvolver um *site* ou aplicação por completo. O momento ideal para aplicá-lo em um projeto de *software* deve, contudo, levar em conta as vantagens e desvantagens mencionadas nesta seção.

2.4.6 Linguagem PHP

O HTML é utilizado para criação de páginas *web*. Porém, sem um modo de manipular o conteúdo destas, o servidor se limita a servir páginas estáticas capazes de exibir somente conteúdo, e que só se modificarão quando o desenvolvedor editar o arquivo HTML e reenviá-lo para o servidor. Para transformar *sites* em aplicações *web* interativas, o servidor precisa de uma ferramenta que lhe dê total controle da comunicação entre o servidor com o cliente, tornando-o capaz de gerar páginas HTML de forma dinâmica e em tempo real (BEIGHLEY; MORRISON, 2010). Para tal é necessário haver uma linguagem *server-side* com tais capacidades.

PHP (acrônimo recursivo para *PHP: Hypertext Preprocessor*, originalmente ***P**ersonal **H**ome **P**age*) é uma linguagem de programação que possui as características comentadas. Trata-se de uma linguagem de *script* de código aberto para uso geral, adequada especialmente para o desenvolvimento *web* e que pode ter seu código fonte embutido dentro do HTML (THE PHP GROUP, 2016). Foi criada em 1994 por Rasmus Lerdorf com propósito de ser uma linguagem *server-side*, isto é, onde o processamento é executado apenas do lado do servidor. O código PHP embutido no HTML será executado sempre que a página correspondente for visitada,

[25] Disponível em: <http://getbootstrap.com/getting-started/#grunt>. Acesso em jan. 2016.

sendo este código interpretado pelo *web server*, que gera um código HTML ou outra saída a ser apresentada ao visitante da página em seu navegador (WELLING; THOMSON, 2005).

Um resumo sobre seu funcionamento é dado na Figura 16. Um navegador *web* (aplicação cliente) faz uma requisição ao servidor para obter um recurso (1). O *script* PHP correspondente é executado no servidor por meio do interpretador PHP (2), podendo este acessar a base de dados ou arquivos por exemplo (3) e alterar ou gerar código HTML de acordo com a requisição recebida (4), retornando o HTML para o usuário (5). O navegador não faz ideia de qual linguagem *server-side* foi utilizada na construção da página correspondente, apenas recebe o HTML estático gerado pelo servidor.

Figura 16 - Funcionamento do PHP

Fonte: Adaptado de Frederico Marinho[26]

Existem outras tecnologias alternativas ao PHP como é o caso de Perl, *Active Server Pages* (ASP), *Java Server Pages* (JSP), Python, Ruby, dentre outras. Porém, segundo Welling e Thomson (2005), se comparado a estas tecnologias o PHP possui algumas vantagens, sendo estas: possui alto desempenho; suporta interfaces para diferentes sistemas de gerenciamento de banco de dados; possui bibliotecas integradas para tarefas comuns da *web*; é gratuito; é fácil de aprender e usar; possui ótimo suporte orientado a objetos; possui portabilidade entre plataformas; disponibilidade de código-fonte aberto; disponibilidade de suporte.

Outro fator importante é que o PHP, atualmente, é a linguagem mais utilizada no desenvolvimento *web*. Segundo uma pesquisa realizada pela W3Techs[27], o PHP é usado por 81,7% de todos os *sites* cuja linguagem de programação do lado do servidor é conhecida. Muito

[26] Disponível em: <http://www.fredericomarinho.com/introducao-ao-desenvolvimento-web-com-php-aula-1-preparando-o-ambiente-para-iniciar-a-programacao/cliente-servidor-php/>. Acesso em jan. 2016.
[27] Disponível em: <http://w3techs.com/technologies/overview/programming_language/all>. Acesso em jan. 2016.

disso é consequência dos gerenciadores de conteúdo, tais como WordPress, Drupal e Joomla. Grande parte dos *blogs* encontrados pela *web* usa alguns destes gerenciadores. Além do mais, o PHP possui a maior comunidade ativa, o que contribui para uma vasta documentação disponível.

Embutir código PHP em páginas HTML é simples e usa o mesmo conceito das *tags* em HTML. A diferença está na sintaxe usada para indicar onde inicia ou termina o código PHP embutido. Existem quatro maneiras diferentes para embutir código PHP no HTML, os quais são mostrados no Quadro 11. Qualquer texto entre as *tags* PHP é interpretado como instruções em PHP, do contrário é tratado como HTML. Além disso, para que o interpretador PHP possa executar os *scripts* inseridos no documento HTML recomenda-se[28] o uso da extensão *.php*.

Quadro 11 - Maneiras de se embutir código PHP no HTML

Estilo	**Sintaxe**
XML	<?php echo ' Olá mundo ! '; ?>
Abreviado	<? echo ' Olá mundo ! '; ?>
Script	<script language='php'> echo ' Olá mundo ! '; </script>
ASP	<% echo ' Olá mundo ! '; %>

Fonte: Elaborado pelo autor

O PHP foi influenciado por linguagens como C, Perl, Java, C++ e *Tool Command Language* (TCL), de tal forma que sua sintaxe é familiar a qualquer desenvolvedor que possua conhecimentos prévios destas linguagens. Independente do histórico de uso, é uma linguagem com boa curva de aprendizado.

Em PHP, as variáveis são declaradas com o caractere inicial "$" (ex: *$resultado*). Além disso, seus identificadores (nomes de variáveis) devem seguir algumas regras, como comenta Welling e Thomson (2005): podem ser de qualquer comprimento e podem consistir em letras, números, sublinhados e sinais de cifrão; não podem iniciar com um dígito; são *case-sensitive*; e podem ter o mesmo nome que uma função, embora recomenda-se evitar tal prática. Uma diferença fundamental entre PHP e outras linguagens é que PHP é uma linguagem fracamente tipada, isto é, o tipo da variável é determinado pelo pelo valor/objeto a ela atribuído.

No quesito segurança, muito se questionou sobre a segurança suportada pelo PHP. De fato, muitas falhas foram identificadas em suas versões iniciais, sendo corrigidas posteriormente. A versão do PHP mais utilizada atualmente é a 5.x, sendo que foi lançado

[28] Há meios de se fazer o interpretador PHP funcionar em páginas com a extensão *.html* ou *.htm*, porém, é recomendado seguir o padrão de salvá-las como *.php*.

recentemente o PHP 7[29] com promessas de melhorias, correções de *bugs* e desempenho superior se comparado a versão anterior.

A criação do PHP permitiu o desenvolvimento de aplicações *web* mais robustas e dinâmicas, possibilitando uma interação mais significativa com o usuário. Trata-se de uma linguagem simples que, ao mesmo tempo, oferece muitos recursos avançados. Com sua rápida popularização, seu uso cresceu aceleradamente, abrangendo a maioria dos *sites* na internet. Isso motivou o desenvolvimento de diversos *frameworks* PHP com o intuito de aumentar a produtividade, segurança e qualidade das aplicações. Na próxima seção serão apresentados alguns destes *frameworks* com suas principais características.

2.4.7 Frameworks PHP

Desenvolver aplicações independente da plataforma pode ser um processo complexo, oneroso e demorado. Neste cenário os *frameworks* podem fornecer um ótimo recurso para facilitar o processo de desenvolvimento pois são ferramentas úteis que fornecem uma estrutura e um ambiente unificado que auxilia no desenvolvimento de forma mais rápida e eficiente. É uma boa prática de desenvolvimento *web* utilizar *frameworks*, que em geral aumentam a escalabilidade e manutenção a longo prazo, de modo que se possa dedicar mais tempo para a construção de código de alta qualidade (AVOYAN, 2015). Existem muitos *frameworks* PHP no mercado, dos quais destacam-se (AVOYAN, 2015; WINSPIRE WEB SOLUTION, 2015):

- **Laravel**: provavelmente um dos mais populares no mercado atualmente. É conhecido por sua elegância e simplicidade. Suporta uma variada gama de necessidades e projetos de programação, do iniciante ao avançado, sendo adequado para projetos de todos os tipos e tamanhos. É construído em cima de vários componentes Symfony que garantem uma estrutura sólida para a produção de código bem testado e confiável. Possui características interessantes, que incluem uma poderosa biblioteca de fila, um *Object Relation Mapping* (ORM, Mapeamento Objeto-Relacional)[30], roteamento fácil e autenticação simples.

[29] Os desenvolvedores decidiram não lançar a versão 6, pois muitas alterações presentes na versão 5.6 já representam o PHP 6.

[30] Trata-se de uma técnica de mapeamento objeto relacional com o objetivo de criar uma camada de mapeamento entre o modelo de objetos (aplicação) e o modelo relacional (banco de dados) de forma a abstrair o acesso ao

- **Symfony**: está nas principais listas de *frameworks PHP*. É construído sobre componentes Symfony, tais como Drupal, Ez Publish, e phpBB. Com mais de 300.000 desenvolvedores a bordo, registram-se cerca 1.000.000 de *downloads* do *framework*. Mais de 1.000 colaboradores ajudam na manutenção do código do Symfony. Ele permite que quase tudo no processo de desenvolvimento seja personalizado, sendo um grande atrativo para os desenvolvedores.
- **CodeIgniter**: adequado para desenvolvedores que desejam criar *sites* e aplicações *web* com recursos completos utilizando para isso um conjunto simples e elegante de ferramentas. É conhecido por ser rápido se comparado com outros *frameworks*, além de ser indicado como o melhor para iniciantes devido a facilidade de aprendizado.
- **Phalcon**: é um *framework PHP* escrito na linguagem C, porém, não há necessidade de aprender a linguagem C, uma vez que suas funcionalidades são expostas à medida que classes PHP estejam prontas para o uso. A vantagem de utilizar C é que o *framework* oferece maior desempenho e independência de plataforma, ou seja, é extensível para Microsoft Windows, GNU/Linux e Mac OS X. Devido a esse desempenho superior, é considerado um dos *frameworks* PHP mais rápido entre os existentes. Além disso, oferece excelentes recursos como traduções, segurança, gestão de *assets*[31], *auto-loader* universal etc.
- **CakePHP**: considerado um *framework* contemporâneo para o desenvolvimento PHP, oferece uma arquitetura flexível para desenvolvimento, manutenção e implementação rápida de aplicações. É carregado com características notáveis, incluindo componentes e *helpers*[32] aprimorados, melhor gerenciamento de sessão, ORM melhorado entre outras. Sua nova versão provê maior modularidade, permitindo criar mais bibliotecas independentes e reduzir o acoplamento.
- **Yii**: é um *framework* PHP baseado em componentes para aplicações *web* que tem crescido em popularidade nos últimos anos. É especialmente conhecido por ser uma plataforma robusta para aplicações *web*. Possui recursos para lidar com segurança, integração com bancos de dados relacionais e não-relacionais, *cache*, autenticação

---

mesmo. Essa abstração é feita de forma que as tabelas do banco de dados são representadas através de classes e os registros como instâncias destas. Mais em: <https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional>. Acesso em jan. 2016.

[31] Termo utilizado para designar tudo o que complementa o conteúdo dos *websites*. Ex: folhas de estilo, scripts, fontes, imagens etc.

[32] Conjunto de funções relacionadas a uma determinada categoria de tarefas, por exemplo, *helpers* de URL para criação de *links*, *helpers* de formulários para criação de *forms* etc.

e controle de acesso por função, testes, dentre outros. Como desvantagem possui curva de aprendizado alta.

- **Zend**: considerado como um dos *frameworks PHP* mais populares para a construção de aplicações *web* de alto desempenho, vem com ferramentas de codificação criptográficas e seguras, as quais permitem executar projetos de desenvolvimento *web* de modo impecável. Inclui características interessantes, como modularidade e extensibilidade. Sua curva de aprendizado também é grande, pois exige conhecimentos avançados em PHP e orientação a objetos.

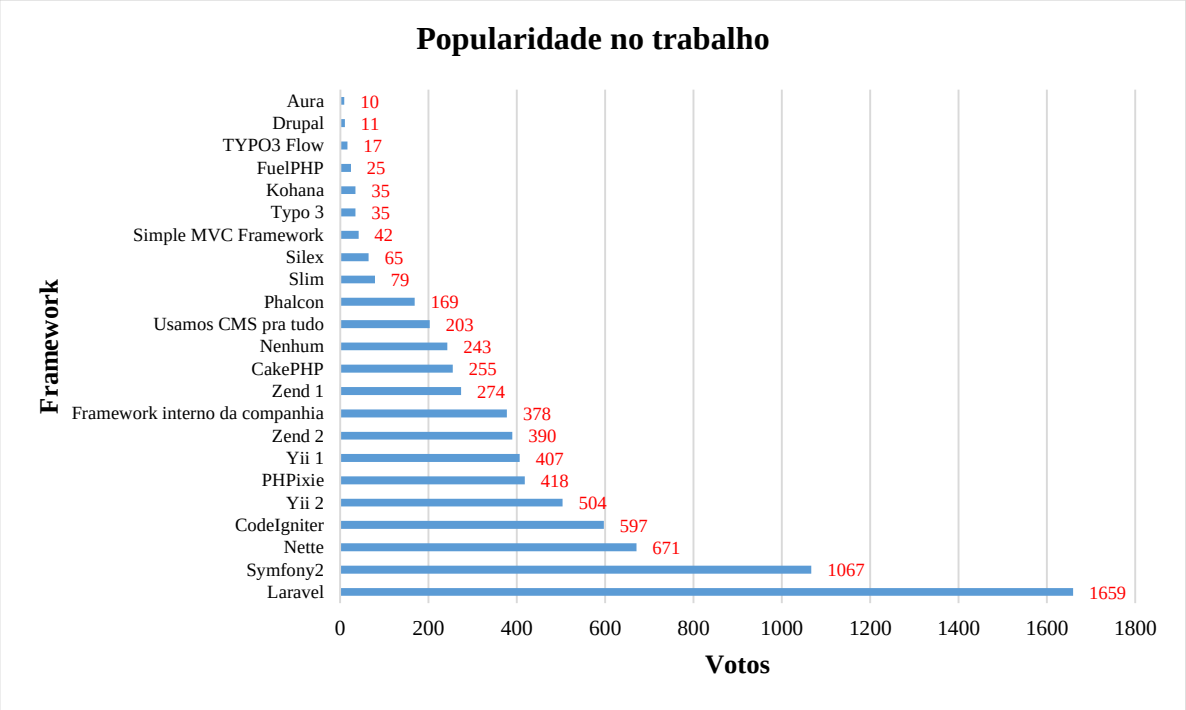
Diante da grande oferta de bons *frameworks*, a escolha de qual utilizar em um projeto de desenvolvimento *web* deve considerar alguns aspectos importantes que podem ter influência na escolha. Dessa forma a escolha deve ser feita considerando:

- Tamanho da comunidade ativa do *framework.*
- Documentação disponível.
- Quantidade de atualizações.
- Curva de aprendizado.
- Quantidade de *plug-ins* disponíveis.
- Recursos oferecidos.
- Adequação ao tipo de aplicação a ser desenvolvida.

O *site* SitePoint[33] realiza anualmente uma pesquisa de popularidade dos *frameworks* PHP. A última realizada no início de 2015 mostra resultados como a popularidade no trabalho e em projetos pessoais. Os Gráficos 1 e 2 apresentam estes resultados.

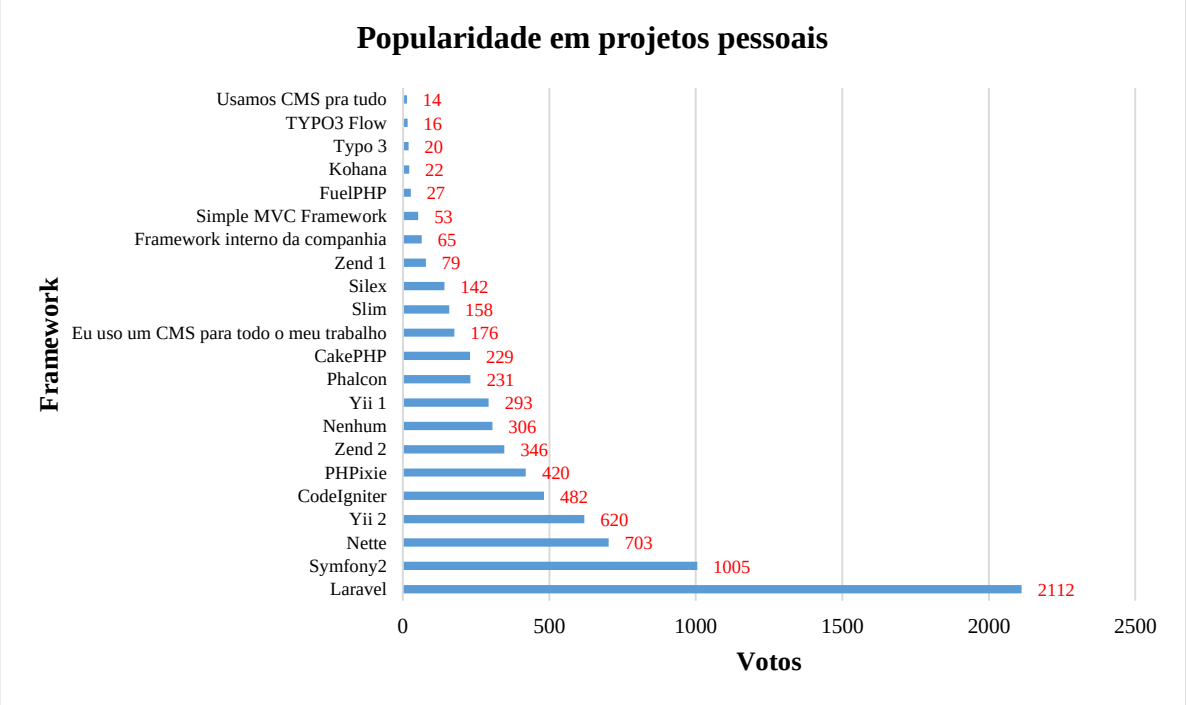
[33] Disponível em: <http://www.sitepoint.com/>. Acesso em jan. 2016.

Gráfico 1 - Popularidade dos *frameworks* PHP no trabalho



Fonte: Adaptado de SitePoint[34]
Os *frameworks* que obtiveram menos que 10 votos foram desconsiderados
Os dados utilizados na pesquisa estão disponíveis em: https://github.com/sitepoint-editors/php-fw-survey-2015

Gráfico 2 - Popularidade dos *frameworks* PHP em projetos pessoais

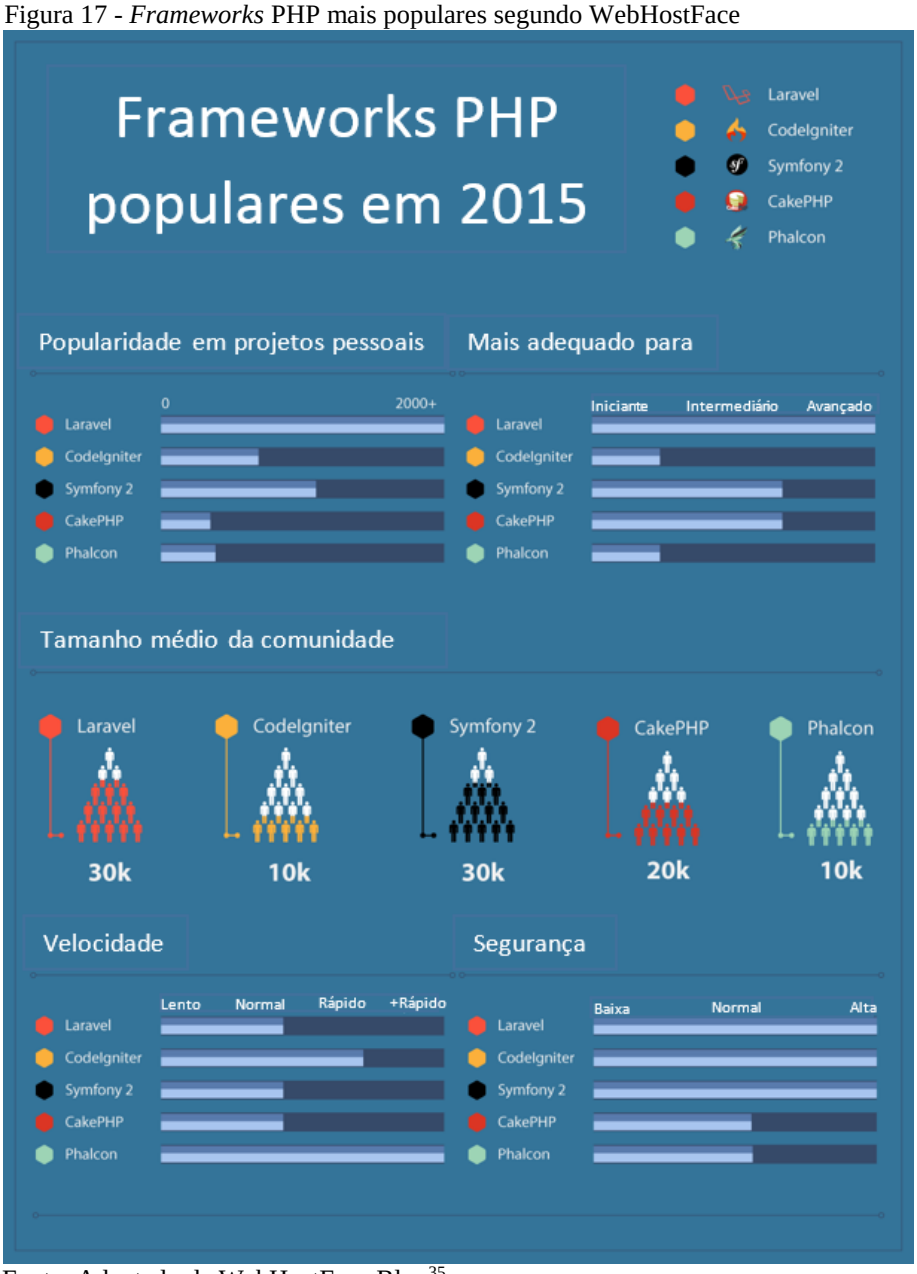


Fonte: Adaptado de SitePoint[34]
Os *frameworks* que obtiveram menos que 10 votos foram desconsiderados
Os dados utilizados na pesquisa estão disponíveis em: https://github.com/sitepoint-editors/php-fw-survey-2015

[34] Disponível em: <http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/>. Acesso em jan. 2016.

Da mesma forma, o *blog* WebHostFace elaborou um infográfico com cinco dos principais *frameworks* PHP, dividindo-os em cinco categorias: (i) popularidade em projetos pessoais, (ii) nível de experiência do usuário, (iii) tamanho médio da comunidade ativa, (iv) velocidade e (v) segurança. O *blog* tomou como base os resultados da pesquisa acima comentada do SitePoint, os projetos PHP mais populares e assistidos do GitHub em 2015, e as principais pesquisas relacionadas no Google Trends. Este trabalho de análise resultou no infográfico apresentado na Figura 17.

Figura 17 - *Frameworks* PHP mais populares segundo WebHostFace



Fonte: Adaptado de WebHostFace Blog[35]

[35] Disponível em: <http://learninglaravel.net/most-popular-php-frameworks-2015-infographic/>. Acesso em jan. 2016

Analisando-se os Gráficos 1 e 2, percebe-se a superioridade do *framework* Laravel em relação aos concorrentes nos quesitos popularidade no ambiente de trabalho e popularidade em projetos pessoais. O infográfico da Figura 17 complementa a análise apresentando resultados em novos quesitos (adequação, tamanho médio, comunidade, velocidade e segurança) comparando os *frameworks* em maior profundidade. Nele é possível observar que dentre os destacados, o Laravel é o único que se adequa a todos os níveis de experiência do usuário, do iniciante ao avançado. Outra informação importante está relacionada com velocidade e segurança - embora o Laravel possua uma velocidade dentro da normalidade, este provê alta segurança. Apesar do Phalcon ser o mais rápido na comparação feita pelo infográfico, em contrapartida não é tão seguro. Embora o CodeIgniter ofereça alta segurança e seja o segundo *framework* mais rápido, é adequado somente para usuários iniciantes, o que leva a imaginar que o mesmo não ofereça recursos tão avançados, e assim como o Phalcon, possui a menor comunidade ativa, de modo que influencia diretamente na documentação disponível e frequência de atualização.

Os resultados apresentados nesta seção não são para evidenciar ou discutir o melhor *framework* PHP, mas para servir como justificativa à escolha feita para implementação *server-side* do sistema Pipou. Assim, com base em um prévio estudo realizado e nos resultados mostrados aqui, optou-se pelo Laravel como *framework* oficial no desenvolvimento deste trabalho.

Vale salientar que todos os *frameworks* PHP supracitados são de código aberto. Além do mais, são implementados com base no padrão *Model-View-Controller* (MVC), ou seja, para utilizá-los corretamente e usufruir de seus recursos por completo a aplicação ou *site web* deve ser construída seguindo o modelo MVC. Por se tratar de um *framework* implementado sobre o MVC, antes de entrar em mais detalhes sobre o Laravel, é preciso entender os conceitos por trás de tal padrão.

2.4.7.1 Modelo MVC

Um dos principais erros na construção de sistemas é quando o controle de acesso a dados, a lógica de negócio e a apresentação são codificados de tal maneira que qualquer alteração em uma destas partes afeta as demais partes do sistema, tornando difícil sua manutenção. Baseado nisso surgiu na década de 70 um padrão de arquitetura desenvolvido para

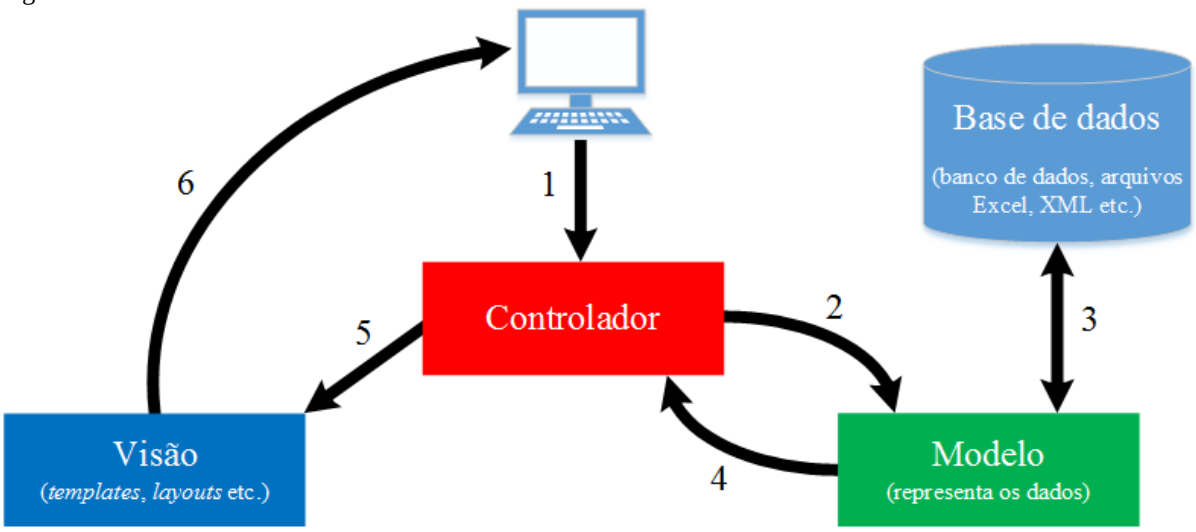
ser utilizado na construção de interfaces com o usuário na linguagem Smalltalk, conhecido como *Model-View-Controller*, comumente cunhado por MVC (GAMMA *et al.*, 2007).

A ideia principal por trás do MVC consiste em dividir uma aplicação de forma organizada em três partes básicas: Modelo, Visão e Controlador. É uma maneira de estruturar uma aplicação de modo que a interface (visão) esteja separada dos dados (modelo), sendo esta separação intermediada por meio dos controladores. Desta forma, o controle de acesso a dados, a lógica de negócio e a apresentação ficam totalmente separados, de modo que cada parte possui suas funções e características bem definidas, facilitando a resolução de um problema maior e promovendo um desenvolvimento muito mais flexível. As camadas do modelo MVC podem ser descritas da seguinte maneira (MARTINS, 2011; DANGAR, 2013, p. 8, tradução do autor; FIGUEIREDO, 2015):

- **Modelo**: responsável por armazenar as regras de negócio da aplicação. São a maneira pela qual pode-se interagir com os dados. É uma camada entre os dados e a aplicação. Pode-se ter validações, acesso ao banco de dados, acesso à arquivos como Excel e XML, cálculos etc.
- **Visão**: é a parte da aplicação que o usuário final vê, isto é, a interface gráfica. Preocupa-se apenas em exibir os dados, não se preocupando em como obtê-los ou quando renderizá-los. Permite aos desenvolvedores escrever a camada de apresentação separada da lógica de negócio.
- **Controlador**: é quem diz “quando as coisas vão acontecer”. É o intermediador entre as camadas de Modelo e Visão. Sua principal responsabilidade é manipular requisições e passar dados do Modelo para a Visão. Não precisa saber como obter os dados (Modelo) nem como exibi-los (Visão), apenas quando fazer isso.

É possível observar na Figura 18 o relacionamento entre as camadas do modelo MVC no contexto do funcionamento de aplicações web. Um usuário faz uma requisição que é capturada pelo servidor através do Controlador (1). O Controlador por sua vez trata a requisição e solicita dados ao Modelo caso necessite (2), o qual acessa os mesmos na base dados (3), aplica filtros ou lógicas se necessário e os retorna para o Controlador (4). Após, o Controlador usa a Visão apropriada (5) para exibir os dados para o usuário (6).

Figura 18 - Relacionamento entre as camadas do modelo MVC



Fonte: Elaborada pelo autor

O modelo MVC é uma ótima arquitetura para o desenvolvimento de aplicações graças às vantagens proporcionadas pelo mesmo. Bucanek (2009) e Marcoratti (2009) destacam: modularidade; flexibilidade; reaproveitamento de código; facilidade de manutenção; facilidade na atualização da interface da aplicação; escalabilidade; simplicidade na inclusão de novos clientes, incluindo apenas seus visualizadores e controladores; facilidade para manter, testar e atualizar sistemas múltiplos, uma vez que o modelo MVC gerencia múltiplas visões; e possibilidade de desenvolvimento em paralelo, pois Modelo, Visão e Controle são independentes. Contudo, o MVC também possui algumas desvantagens, sendo estas (MARCORATTI, 2009): requer mais tempo para análise e modelagem do sistema; não é trivial, requer pessoal especializado para sua modelagem; não é aconselhável para aplicações de pequeno porte.

Por fim, o padrão MVC foi originalmente desenvolvido para sistemas *desktop*, mas se popularizou amplamente para aplicações *web*. Assim como o Laravel, inúmeros *frameworks* foram implementados baseados neste modelo, diferindo no modo como as responsabilidades são assumidas entre cliente e servidor, mantendo, contudo, a ideia original proposta pelo padrão.

2.4.8 Framework Laravel

Em 2009, o panorama dos *frameworks* consistia principalmente de Symfony, Zend, Slim Micro *Framework*, Kohana, Lithium e CodeIgniter. Devido a sua grande comunidade, simplicidade e documentação compreensiva, o CodeIgniter era provavelmente o *framework* PHP mais conhecido e utilizado naqueles tempos. No entanto, Taylor Otwell em 2011 viu algumas funcionalidades essenciais para o desenvolvimento *web* ausentes no CodeIgniter, como suporte embutido para autenticação e uso de *clousures* - função anônima que pode ser atribuída a uma variável ou passada para uma outra função como um parâmetro - no roteamento. Pensando nisso criou o *framework* Laravel, e em junho de 2011 lançou a primeira versão beta do mesmo. A sua primeira versão contava com diversas funcionalidades interessantes como autenticação, ORM *Eloquent*, localização, modelos e relações, roteamento simples, *caching*, sessões, *views*, extensibilidade por meio de módulos e bibliotecas, *helpers* para *forms* e HTML, métodos de validação, paginação, um instalador de pacote de linha de comando, entre outras. Neste ponto, o Laravel ainda não era um *framework* MVC pois não dispunha de controladores, porém sua sintaxe limpa e seu grande potencial fizeram que diversos desenvolvedores começassem a adotá-lo. A rápida popularização do mesmo motivou o lançamento da segunda versão seis meses após seu lançamento, a qual incluiu controladores passando a ser reconhecido oficialmente como um *framework* MVC (SURGUY, 2013). O Laravel se tornou um dos *frameworks* mais populares no mundo, e se encontra atualmente em sua versão 5.2.

Para se instalar o Laravel é necessário o uso do Composer utilizado para gerenciar as dependências do *framework*. Instalado o Composer, uma das formas de instalação do Laravel é realizar o *download* do *framework* e, após realizar sua extração no local desejado, navegar até o diretório por uma interface de linha de comando (ex: *prompt* de comando do Windows) e executar o comando *composer install*, que instala automaticamente o Laravel e suas dependências com base no arquivo *composer.json* presente na pasta do *framework*. Mais detalhes sobre a instalação e configurações específicas do Laravel podem ser encontrados em https://laravel.com/docs/4.2.

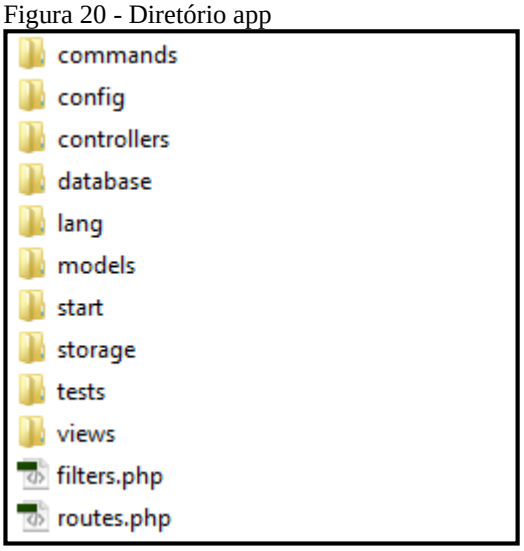
A estrutura de arquivos do *framework* possui várias pastas e arquivos, sendo a pasta *app* a mais importante pois é nela que a aplicação irá residir. É o local onde as configurações, modelos, controladores (seção 2.4.8.3), visões, *migrations* (seção 2.4.8.8), *seeds* (seção 2.4.8.9) entre outros estarão. Na pasta *public* é onde as imagens, JavaScripts e CSSs serão hospedadas para que fiquem acessíveis no navegador. Mais explicações sobre cada arquivo e diretório do

Laravel podem ser consultadas em (REES, 2013). As Figuras 19 e 20 mostram a estrutura de arquivos do Laravel e o conteúdo do diretório *app*, respectivamente, após instalação do *framework*.

Figura 19 - Estrutura de arquivos do Laravel

Fonte: Elaborada pelo autor

Figura 20 - Diretório app



Fonte: Elaborada pelo autor

O Laravel incentiva o uso de boas práticas de programação e utiliza o padrão *PHP Standards Recommendation 2* (PSR-2) como guia para estilo de codificação. Assim, tem como principal característica ajudar a desenvolver aplicações seguras e performáticas de forma rápida, com código limpo e simples. Visto a quantidade de recursos fornecidos pelo *framework*, serão abordados resumidamente nesta seção aqueles mais utilizados no desenvolvimento deste trabalho, que são: rotas, filtros, *controllers*, *blade*, validação, *schema builder*, *migrations*, *seeding* e o ORM *Eloquent*.

2.4.8.1 Rotas

Toda requisição feita ao servidor é tratada internamente por um mecanismo chamado *Routing*, ou simplesmente Sistema de rotas, que é gerenciado pelo Laravel através do arquivo *routes.php* (FIGURA 20) de uma forma bastante simples. Para exemplificar, seja o caso onde um usuário acessa o endereço *http://exemploderota.com/minhapagina*. Uma requisição do tipo *GET* é feita para o servidor para obter o recurso informado no *Uniform Resource Locator* (URL,

Localizador Padrão de Recursos). Esta requisição é capturada no Laravel, sendo que uma das formas de tratá-la no arquivo de rotas é dada no Quadro 12.

Quadro 12 - Tratamento de requisição *GET* pelo sistema de rotas do Laravel

```
Route::get('minhapagina', function() {
    return 'Olá mundo!';
}
```

Fonte: Elaborado pelo autor

Os verbos HTTP disponíveis no Laravel são ***GET***, ***POST***, ***PUT***, ***DELETE***, todos sendo utilizados conforme o modelo apresentado no Quadro 12. O *framework* ainda disponibiliza o método *Route::any()* que é utilizado para capturar qualquer tipo de requisição sem diferenciação, porém recomenda-se utilizar o verbo HTTP correto para cada situação, como explica Rees (2013).

Outra funcionalidade suportada pelo sistema de rotas do Laravel é a passagem de parâmetros para rotas usados para inserir *placeholders*[36] nas definições de rotas, criando um espaço em cada segmento que pode ser capturado dentro da *closure* e passado para a lógica da aplicação. O Quadro 13 mostra um exemplo para melhor entendimento. Neste exemplo, o *placeholder {genre}* mapeará qualquer coisa que seja digitada após o URL identificado por */books/*. O Laravel passará o valor para o parâmetro *$genre* da *closure*, permitindo utilizá-lo dentro da lógica. O uso do parâmetro pode ser opcional, adicionando um ponto de interrogação no final do seu nome (ex: *{genre?}*). Caso nenhum gênero seja fornecido com a URL, o valor da variável *$genre* será nulo. Toda requisição espera uma resposta. O sistema de roteamento do Laravel fornece diversos tipos de respostas, como *strings*, *views* (páginas HTML), redirecionamentos, respostas personalizadas, JSON e *download.*

Quadro 13 - Passagem de parâmetros para rotas no Laravel

```
Route::get('/books/{genre}', function($genre) {
    return 'Books in the {$genre} category';
}
```

Fonte: (REES, 2013, p. 56-57)

[36] Em português, espaço reservado. É aquilo que mantém, denota ou reserva um lugar para algo que virá depois e o substituirá.

2.4.8.2 Filtros

Segundo Rees (2013, p. 71, tradução do autor), filtros são "determinados conjuntos de regras que podem ser aplicadas em uma rota". Tais regras podem ser aplicadas antes ou depois da lógica de uma rota ser executada. O Laravel disponibiliza um arquivo padrão para implementação de filtros, chamado *filters.php* (FIGURA 20), mas nada impede o usuário de implementá-los onde quiser.

Para criar um filtro, usa-se o método *Route::filter()* cujo primeiro parâmetro é o nome do filtro e o segundo é uma função de retorno chamada somente quando o filtro é executado. Para relacionar um filtro a uma rota utiliza-se um vetor como segundo parâmetro da rota, o qual permite atribuir parâmetros adicionais. O Quadro 14 exemplifica a criação de um filtro e sua atribuição a uma rota. Neste exemplo, se a data atual for igual a '12/12/84', a *clousure* retornará uma resposta. Se uma resposta for retornada da *closure*, então ela que será servida ao invés da lógica da rota - no caso do presente exemplo o usuário será direcionado para a *view* de feliz aniversário enquanto as lógicas das demais rotas seguem seu fluxo normal. O índice *'before'* diz ao *framework* que o filtro *'birthday'* deve ser executado antes da lógica da rota.

Quadro 14 - Exemplo de criação de filtro e atribuição à rota

```
// criação do filtro
Route::filter('birthday', function() {
    if (date('d/m/y') == '12/12/84') {
        return View::make('birthday');
    }
});

// atribuição do filtro a uma rota
Route::get('/', array(
    'before' => 'birthday',
    function() {
        return View::make('hello');
    }
));
```

Fonte: (REES, 2013, p. 72-74)

Muito pode-se fazer com filtros no Laravel. É uma ótima forma de proteger o acesso a determinadas rotas e prover mais segurança para a aplicação. Mais sobre filtros em (REES, 2013; OTWELL, 2016).

2.4.8.3 *Controllers*

Como o Laravel segue o padrão MVC o tratamento de rotas é feito, naturalmente, com uso de *controllers*. O *controller* consiste em uma classe PHP utilizada para armazenar a lógica de uma rota que contêm métodos públicos conhecidos como ações. As ações podem ser imaginadas como uma alternativa para as *clousures* utilizadas nas rotas (REES, 2013). Toda classe *controller* deve ser armazenada no diretório *app/controllers* (FIGURA 20). O Quadro 15 mostra um exemplo de implementação de um *controller*.

Quadro 15 - Exemplo de implementação de um *controller*

```
<?php

class ArticleController extends BaseController {
    public function showIndex() {
        return View::make('index');
    }

    public function showSingle($articleId) {
        return View::make('single');
    }
}
```

Fonte: (REES, 2013, p. 86)

Para utilizar *controllers* nas rotas, é preciso fazer a conexão entre um *Uniform Resource Identifier* (URI, Identificador Uniforme de Recursos) e o *controller*. Seja o método *Route:get()*. Por ele cria-se uma nova rota, composta agora por dois parâmetros: o primeiro é o mesmo já utilizado, porém, o segundo não se trata mais de uma *closure*, e sim de uma *string* composta por duas seções separadas pelo caractere '@'. A primeira seção consiste no nome do *controller*, a segunda é o nome do método a ser invocado.

O Laravel também oferece o roteamento de *controllers* por meio do padrão RESTful[37]. Este mecanismo permite o roteamento de qualquer um dos verbos HTTP supracitados em uma única rota, eliminando a necessidade de criação de rotas individuais. A sintaxe da rota segue o mesmo conceito anterior, porém, não se utiliza mais os verbos HTTP na chamada do método

[37] É um estilo de arquitetura para comunicação baseada na *web* que permite aos clientes conversar com os servidores de maneira única. No centro de uma arquitetura RESTful está o conjunto de recursos. Tais recursos são identificados por URIs (como um URL) e uma representação interna (uma forma de dados autodescritivos). Por fim, há um conjunto de operações através das quais é possível manipular os recursos (JONES, 2012).

da rota. O segundo parâmetro é somente o nome do *controller*. O Quadro 16 apresenta um exemplo diferenciando estes tipos de roteamento de *controllers*.

Quadro 16 - Exemplos de roteamento de *controllers*

```
// roteamento padrão do controller
Route::get('index, 'ArticleController@showIndex');

// roteamento por meio do padrão RESTful
Route::controller('produtos, 'ProdutoController');
```

Fonte: Elaborado pelo autor

Para utilização do padrão RESTful é necessário alterar os nomes dos métodos implementados no *controller*. A rota é tratada conforme o prefixo dado aos nomes dos métodos no *controller*, que devem obrigatoriamente começar com algum dos verbos HTTP (por exemplo, *public function getRegistros() { }*). Desta forma o Laravel executa o método automaticamente de acordo com o tipo da requisição. Seja o exemplo de roteamento apresentado no Quadro 16. Caso o usuário acesse a URL *produtos/registros* por meio de uma requisição do tipo *GET*, o Laravel busca no *controller ProdutoController* o método com o nome *getRegistros* e o executa. É importante lembrar que neste padrão os nomes dos métodos devem sempre ser prefixados com um dos verbos HTTP, seguido por outro nome com a primeira letra maiúscula (formato *camelCase*).

2.4.8.4 *Blade*

Blade é o sistema de *templates* parte componente do Laravel que permite escrever *views* (páginas HTML) de maneira diferente do convencional. Não utiliza *tags* PHP diretamente no HTML, mas sim outra sintaxe com chaves e arrobas. Uma *view* é armazenada por padrão no diretório *app/views* do Laravel (FIGURA 20), e quando escritas utilizando o *template Blade* devem ser salvas com a extensão *.blade.php* para instruir o Laravel que nestas *views* são utilizadas estruturas do *Blade*. Um ponto importante é que o *Blade* não impede que a sintaxe de PHP puro seja utilizada; na prática o Laravel converte todos os códigos *Blade* em código PHP de forma que o *template* apenas simplifica a escrita do código (DIAS, 2014b).

Na sintaxe do *Blade*, todo código que estiver contido entre {{ duas chaves }} é convertido em um "echo" quando o *template* correspondente é processado. Isso torna a sintaxe mais limpa e fácil de escrever. Nos casos onde seja preciso escapar valores coloca-se o conteúdo

entre {{{ três chaves }}} de forma que caracteres especiais sejam convertidos em entidades HTML.

O *Blade* também simplifica a codificação de estruturas de controle e repetição como *if*, *while*, *for* e *foreach*. Não é preciso digitar *tags* de abertura e fechamento do código PHP, basta prefixar a estrutura de controle com o caractere "@", como mostra o código exemplo no Quadro 17. No *Blade* todas as estruturas de controle e *helpers* utilizam o caractere arroba como prefixo.

Quadro 17 - Exemplo de estrutura de controle no *Blade*

```
@for ($i = 0; $i < 10; $i++)
    <p>Imprimiu número {{ $i }}.</p>
@endfor
```

Fonte: Elaborado pelo autor

Uma grande vantagem do *Blade* é que o mesmo oferece uma maneira de montar *templates* que se beneficia do mecanismo de herança de *templates*, recurso este útil quando várias páginas possuem parte do conteúdo em comum, evitando repetição de código. São exemplos desta situação elementos como menus, barras laterais, cabeçalhos e rodapés, que normalmente não se alteram entre as páginas do *site* tendo, portanto, código idêntico. Para utilizar este recurso do *Blade*, antes é preciso criar um *layout* padrão, o qual será utilizado em todas as páginas. Dentro deste *layout* definem-se áreas onde o conteúdo das páginas herdeiras será inserido. O Quadro 18 mostra um exemplo de *layout* com herança.

Quadro 18 - Exemplo de *layout* a ser herdado

```
<!doctype html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <title></title>
        @section('head')
            <link rel="stylesheet" href="style.css" />
        @show
    </head>
    <body>
        @yield('body')
    </body>
</html>
```

Fonte: (REES, 2013, p. 98)

A declaração *@yield('body')* instrui o *Blade* para criar uma seção a qual poderá ser preenchida posteriormente com conteúdo. A *string* entre os parênteses é o apelido para a seção, usado para referenciá-la. Entre as *tags @section* e *@show* pode-se definir algum conteúdo padrão que é exibido por *default*, a menos que o *template* herdeiro decida reescrevê-lo (REES,

2013). Com os *templates Blade* é possível implementar tantos níveis de herança quanto necessário. Um exemplo de *template* herdeiro pode ser conferido no Quadro 19.

Quadro 19 - Exemplo de *template* herdeiro

```
{{-- diz ao Blade qual o template pai --}}
@extends('layouts.base')
{{-- tudo entre @section e @stop será inserido dentro do template pai --}}
@section('head')
    {{-- diz ao Blade para adicionar o conteúdo padrão da seção pai sem substituí-lo --}}
    @parent
    {{-- acrescenta conteúdo à seção 'head' do template pai --}}
    <link rel="stylesheet" href="another.css" />
@stop

{{-- este conteúdo será inserido no template pai onde a declaração @yield('body') está --}}
@section('body')
    <h1>Hurray!</h1>
    <p>We have a template!</p>
@stop
```

Fonte: Adaptado de (REES, 2013, p. 101)

2.4.8.5 Validação

O mecanismo de validação é utilizado em aplicações para garantir que os dados recebidos são corretos, evitando gerar inconsistências no banco de dados ou até mesmo vulnerabilidade na segurança da aplicação. O Laravel possui uma enorme quantidade de regras de validação nativas para uso, que podem ser declaradas em um vetor seguindo a seguinte sintaxe: *array('<nome_do_campo>' => '<nome_da_regra>')*[38]. A lista com todas as regras de validação disponíveis pode ser consultada na documentação oficial, disponível em Otwell (2016).

Para usar a validação no Laravel, basta: obter os dados recebidos da aplicação cliente, montar o vetor com as regras de validação necessárias e utilizar o objeto *Validator* do Laravel para validar os campos. O trecho de código apresentado no Quadro 20 ilustra o uso de um *Validator* para aplicar aos campos obtidos da requisição uma validação que garanta que os valores neles contidos sejam alfanuméricos, com tamanho máximo de 30 caracteres.

[38] O termo "array()" em PHP pode ser substituído por "[ ]".

Quadro 20 - Exemplo de validação

```
Route::post('/registra', function() {
    // obtém todos os dados da requisição
    $campos = Input::all();
    // constrói o vetor de regras de validação a serem aplicadas
    $regras = array(
        'username' => 'alpha_num|max:30'
    );

    // cria uma instancia do objeto validador
    $validador = Validator::make($campos, $regras);

    // verifica se os campos passaram na validação
    if ($validador->passes()) {
        // Aqui normalmente se faz alguma coisa com os dados
        return 'Os dados foram salvos';
    }

    // caso a validação falhe, redireciona para a raiz da aplicação
    return Redirect::to('/');
});
```

Fonte: Adaptado de (REES, 2013, p. 178-179)

Quando nenhuma das regras do Laravel se aplicam a uma determinada validação, o *framework* permite a criação de regras personalizadas, construídas através do método *Validator::extend()* que recebe dois parâmetros: nome da regra e uma *clousure* para validar o dado, que deve retornar *true* ou *false*. A sintaxe de uso da regra personalizada segue o mesmo padrão das regras nativas. O Quadro 21 mostra um exemplo de criação de uma regra de validação personalizada.

Quadro 21 - Exemplo de criação de regra personalizada

```
Validator::extend('awesome', function($nomeCampo, $valor, $parametros) {
    return $valor == 'awesome';
});
```

Fonte: Adaptado de (REES, 2013, p. 202)

2.4.8.6 *Schema Builder*

O *Schema Builder* (construtor de esquema) é um mecanismo fornecido pelo Laravel para permitir a manipulação de tabelas no banco de dados por meio da classe *Schema*. Funciona muito bem com todas as bases de dados suportadas pelo *framework*, e tem uma API unificada

em todos estes sistemas (OTWELL, 2016). Para criar uma tabela por exemplo, usa-se o método *Schema::create()*, o qual recebe dois parâmetros: nome da tabela e uma *clousure* com um único parâmetro que será usado para criar a estrutura da tabela. Um exemplo da sintaxe de criação de uma tabela por este mecanismo pode ser visualizado no Quadro 22.

Quadro 22 - Sintaxe de criação de tabela pelo *Schema Builder*

```
Schema::create('usuario', function($table) {
    // chave primária auto incremento
    $table->increments('id');

    // colunas do tipo varchar (segundo parâmetro é quantidade máxima de caracteres)
    $table->string('login', 45);
    $table->string('email', 150);
    $table->string('password', 20);
});
```

Fonte: Elaborado pelo autor

O Laravel fornece diversos tipos de colunas que são criadas através dos métodos disponíveis no objeto informado como parâmetro para a *clousure*, normalmente chamado de *$table*. A classe *Schema* oferece ainda outros métodos para manipulação de tabelas, como o *rename()*, que renomeia uma tabela, e o *table()*, que altera propriedades de uma tabela existente. Este último recebe os mesmos parâmetros do método *create()*. O método *table()* permite executar ações como criação de novas colunas, remoção e renomeação de colunas, remoção do atributo chave primária, adição e remoção de índices etc. Mais detalhes sobre todos os tipos e modificadores de colunas disponibilizados pelo Laravel e suas respectivas descrições podem ser encontrados em Otwell (2016).

2.4.8.7 *Migrations*

As *Migrations* (migrações) são *scripts* PHP usados para mudar a forma como se estrutura a base de dados. Consistem em um tipo de controle de versão para o banco de dados que permite uma equipe modificar o esquema do banco de dados e manter-se atualizada sobre o estado atual deste. Elas têm hora marcada de criação, sendo executadas sempre na ordem correta, mantendo a estrutura do esquema consistente. O Laravel mantém um registro de quais migrações já foram executadas dentro de uma outra tabela na sua conexão padrão de base de dados. Assim, apenas novas migrações são executadas. Migrações são normalmente

emparelhadas com o *Schema Builder* para gerenciar facilmente o banco de dados da aplicação (REES, 2013; OTWELL, 2016).

O Laravel utiliza uma tabela específica no banco de dados para controlar o *status* de suas migrações. A configuração do nome desta tabela fica dentro do arquivo *app/config/database.php*. Por padrão o Laravel fornece um nome. Para instalar a tabela de migrações basta rodar o comando do Artisan *php artisan migrate:install*.

Para criar uma migração, executa-se o comando: *php artisan migrate:make nome_da_migracao*. O Laravel gera automaticamente um *template* para uma nova migração localizado dentro do diretório *app/data/migrations* (FIGURA 20). Este *template* será nomeado com o parâmetro fornecido no comando prefixado pela data e hora de criação do arquivo (ex: *2015_04_03_135340_create_database.php*). O Quadro 23 mostra o exemplo de um *template* de migração criado pelo Laravel.

Quadro 23 - *Template* de uma migração

```
<?php

use Illuminate\Database\Migrations\Migration;

class CreateUsers extends Migration {
    // roda a migration
    public function up() {
    //
    }

    // desfaz a migration
    public function down() {
    //
    }
}
```

Fonte: Elaborado pelo autor

Todos os comandos e métodos fornecidos pelo *Schema Builder* podem ser utilizados dentro dos métodos *up()* e *down()*. Tudo que é feito no método *up()* deve ser desfeito no método *down()*.

Para rodar as migrações criadas pela primeira vez, utiliza-se o comando *php artisan migrate*. A saída do comando é uma lista com o nome de todas as migrações que foram executadas. Por outro lado, se por algum motivo algum dos arquivos de migração precisa ser alterado, executa-se o comando *php artisan migrate:refresh* para reverter todas as migrações e rodá-las novamente. Por fim, para reverter as alterações aplicadas por uma migração, basta utilizar o comando *php artisan migrate::rollback*. Este comando reverte apenas as migrações

que foram executadas na última vez que o comando *migrate* foi usado. Caso deseje-se reverter todas as migrações, o comando *php artisan migrate::reset* deve ser utilizado.

2.4.8.8 *Seeding*

O Laravel também inclui uma maneira simples para semear o banco de dados com dados de teste usando classes específicas para isso, conhecidas como "classes de sementes". Todas estas classes são armazenadas em *app/database/seeds* (FIGURA 20). Por padrão, uma classe *DatabaseSeeder* é definida pelo Laravel. A partir dela, pode-se usar o método de chamada para executar outras classes de sementes, o que lhe permite controlar a ordem de semeação. O Quadro 24 apresenta um exemplo de implementação básica para utilização do recurso de *seeding* do Laravel.

Quadro 24 - Exemplo de implementação de classes para semear um banco de dados

```
// Classe semeadora
class DatabaseSeeder extends Seeder {
    // Chama a execução das classes sementes
    public function run() {
        $this->call('EstadoCivilSeeder');
    }
}

// Classe semente
class EstadoCivilSeeder extends Seeder {
    // Semeia o banco de dados com os estados civis
    public function run() {
        $dados = ['Solteiro(a)', 'Casado(a)', 'Separado(a)', 'Divorciado(a)', 'Viúvo(a)'];

        foreach ($dados as $d) {
            $ec = new EstadoCivil();
            $ec->nome = $d;
            $ec->save();
        }
    }
}
```

Fonte: Elaborado pelo autor

Para semear o banco de dados, basta digitar o comando do Artisan *php artisan db::seed*. Por padrão, este comando irá executar a classe *DatabaseSeeder*. Caso seja necessário a execução de uma classe em particular, usa-se o comando *php artisan db::seed* --

class=nome_da_classe. É possível também semear o banco em conjunto com as migrações. Por exemplo, o comando *php artisan migrate --seed* roda todas as migrações e em seguida semeia o banco de dados.

2.4.8.9 ORM *Eloquent*

O ORM *Eloquent* consiste em um dos recursos mais poderosos fornecidos pelo *framework* Laravel que permite que objetos sejam instanciados e seus atributos preenchidos com os dados e salvos diretamente no banco de dados de uma forma bastante simples e intuitiva, sem a necessidade de comandos em *Structured Query Language* (SQL). O *Eloquent* é uma implementação do padrão de projeto *Active Record*[39] que consiste de uma classe composta por diversos métodos para interação com o banco de dados (*model*), sendo para tal necessário apenas estender a classe *Eloquent* (DIAS, 2014a). As classes modelos são armazenadas no diretório *app/models* (FIGURA 20). O Quadro 25 apresenta um exemplo de *model* para mapear uma tabela do banco de dados.

Quadro 25 - Exemplo de *model* básico para utilização do *Eloquent*

```
<?php

class Game extends Eloquent {

}
```

Fonte: (REES, 2013, p. 264)

O Laravel segue uma série de padrões que podem ser alterados dependendo do caso ou vontade do usuário. Por exemplo, o *Eloquent* adota alguns padrões para realizar o mapeamento das classes nas tabelas. Um deles é a nomenclatura, onde o Laravel busca pela tabela com o mesmo nome da classe acrescido de "s" (DIAS, 2014a). No caso do exemplo do Quadro 25, *games*. Caso seja preciso definir um nome diferente para a tabela, é necessário adicionar o atributo *$table* ao modelo. Esta e outras modificações podem ser encontradas em (REES, 2013).

Seja o exemplo dado no Quadro 26 que apresenta um trecho de código com objetivo de salvar um novo registro na tabela *games*. O armazenamento persistente no banco de dados do

[39] Trata-se de um padrão de projeto encontrado em aplicações que armazenam seus dados em banco de dados relacionais. A interface de um objeto deve incluir funções como por exemplo *insert*, *update*, *delete* e propriedades que correspondam diretamente às colunas do banco

objeto com os campos nome e descrição é feito pelo trecho de código apresentado no Quadro 26.

Quadro 26 – Inserção de novo registro com o *Eloquent*

```
$game = new Game();
$game->nome = 'Assassins Creed';
$game->descrição = 'Assassins VS templars.';

// salva uma nova tupla no banco
$game->save();
```

Fonte: Adaptado de (REES, 2013, p. 264)

Para atualizar um registro, utiliza-se o mesmo método *save()* usado na inserção com a diferença que o registro precisa ser carregado no banco de dados para posterior modificação. O *Eloquent* fornece o método *find()*, o qual busca e retorna um registro no banco baseado na chave primária informada como parâmetro. No caso de remoção, o conceito para buscar é mesmo, porém não se utiliza o método *save()*, simplesmente usa-se o método *delete()* para apagar o registro do banco. Um exemplo de alteração e remoção podem ser observados no código exibido pelo 7.

Quadro 27 – Atualização e Remoção com o *Eloquent*

```
/*** PARA ALTERAR UM REGISTRO ***/
$game = Game:: find(1);
$game->nome = 'Assassins Creed 4';
$game->descrição = 'Shiver me timbres, Edward.';

// salva alterações do registro no banco
$game->save();
/************************************/

/*** PARA REMOVER UM REGISTRO ***/
$game = Game:: find(1);

// remove registro no banco
$game->delete();
/************************************/
```

Fonte: Adaptado de (REES, 2013, p. 273)

Além dos métodos *find()*, *save()* e *delete()* o ORM *Eloquent* fornece diversos outros métodos para a realização de buscas específicas no banco de dados, os quais podem ser vistos com maiores detalhes em Otwell (2016). Caso ainda o desenvolvedor não se sinta confortável em utilizar o ORM, o Laravel oferece como alternativa o *Query Builder* (construtor de consulta) que fornece uma interface para criar e executar consultas de banco de dados. Ele pode ser usado

para realizar a maioria das operações de banco de dados na aplicação e funciona em todos os sistemas de banco de dados suportados. Além disso, ainda provê um método para a criação de *RAW expressions* (expressões "crus"), que são expressões codificadas em SQL puro (OTWELL, 2016). Exemplos de uso do *Query Builder* e do método para criação de *raw expressions* podem ser vistos no Quadro 28.

Quadro 28 - Exemplo de uso do *Query Builder*

```
// Query Builder
// Isso será convertido pelo Laravel em SELECT * FROM produto WHERE preco > 100
$produtos = DB:: table('produto')->where('preco', '>', 100)->get();

// RAW expression.
$produtos = DB:: select(DB:: raw(SELECT * FROM produto WHERE preco > 100));
```

Fonte: Elaborado pelo autor

Por fim, é provável que a base de dados de uma aplicação possua tabelas que se relacionam umas com as outras. O *Eloquent* facilita o gerenciamento e trabalho com estas relações oferecendo suporte para muitos tipos de relacionamento dos quais se destacam: *One To One* (Um-pra-Um), *One To Many* (Um-pra-Muitos) e *Many To Many* (Muitos-pra-Muitos). Os Quadros 29, 30 e 31 apresentam exemplos da criação destes relacionamentos pelo *Eloquent*.

Quadro 29 - Exemplo de relação Um-pra-Um

```
class Pessoa extends Eloquent {
    // Uma Pessoa tem um Servidor atrelado a ela.
    public function servidor() {
    /* Caso a chave estrangeira na tabela de cidades seja nomeada fora dos padrões, é
    possível passar o nome da coluna como segundo parâmetro do método */
        return $this->hasOne('Servidor', 'pes_id');
    }
}

// Definindo a relação inversa
class Servidor extends Eloquent {
    // Um Servidor É uma Pessoa
    public function pessoa() {
    /* Como a chave estrangeira está na tabela representada por este modelo, utiliza-se
    este método para afirmar que o modelo Servidor é relacionado ao modelo Pessoa */
        return $this->belongsTo('Pessoa', 'pes_id');
    }
}
```

Fonte: Elaborado pelo autor

Quadro 30 - Exemplo de relação Um-pra-Muitos (continua)

```
class Estado extends Eloquent {
    // Um Estado tem muitas Cidades
    public function cidades() {
```

Quadro 30 - Exemplo de relação Um-pra-Muitos (conclusão)

```
        return $this->hasMany('Cidade', 'est_id');
    }
}

// Definindo a relação inversa
class Cidade extends Eloquent {
    // Uma Cidade pertence a um Estado
    public function estado() {
        return $this->belongsTo('Estado', 'est_id');
    }
}
```

Fonte: Elaborado pelo autor

Quadro 31 - Exemplo de relação Muitos-pra-Muitos

```
class Autor extends Eloquent {
    // Um Autor tem muitos Livros
    public function livros() {
    /* Informa ao Laravel que deve verificar a tabela de integração para buscar os modelos.
    É possível passar mais três parâmetros opcionais: nome da tabela de integração,
    primeira chave estrangeira, segunda chave estrangeira */
        return $this->belogsToMany('Livro, 'autor_livro', 'aut_id', 'liv_id');
    }
}

class Livro extends Eloquent {
    // Um Livro tem muitos Autores
    public function autores() {
        return $this->belogsToMany('Autor, 'autor_livro', 'liv_id', 'aut_id');
    }
}
```

Fonte: Elaborado pelo autor

2.4.9 Design Web responsivo

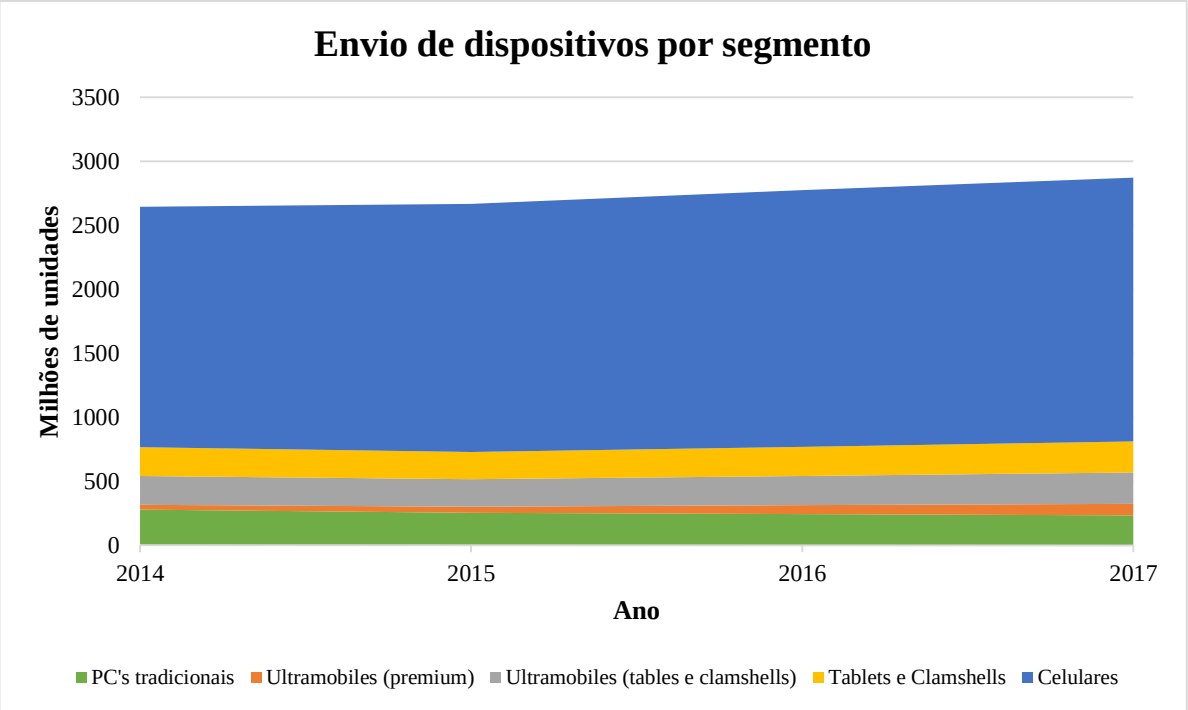
Com a popularização dos celulares, *smartphones* e *tablets* o acesso às informações e a comunicação tornaram-se mais práticas. A evolução das tecnologias de comunicação tais como *wireless*, internet móvel (3G, 4G) e *bluetooth*, bem como outros recursos que tais dispositivos possuem, estão dando credibilidade em sua utilização em diversas situações antes dominadas por computadores portáteis e de mesa, pois devido ao seu tamanho reduzido torna-se mais fácil estar com o dispositivo sempre presente, seja para o lazer ou trabalho, permitindo gerenciar

melhor os procedimentos e tarefas do dia a dia com uso destas plataformas móveis (DA ROSA; DA SILVA, 2013).

O tráfego de dados móveis tem aumentado cada vez mais. Segundo a edição de 2015 do *Visual Networking Index* (VNI, Índice Visual de Redes) da Cisco[40], em 2014 cerca de 88% dos dados trafegaram de modo 'inteligente' (capacidades avançadas de computação/multimídia, com velocidade mínima de 3G) com volume de 30 bilhões de gigabytes (GB), sendo que até 2019 esse percentual deverá chegar a 97%, alcançando volume anual de 292 bilhões de GB. Os principais motivos desse crescimento estão relacionados à adoção contínua de dispositivos móveis.

Uma pesquisa realizada pelo Instituto Gartner (importante empresa de consultoria e pesquisa de tecnologia da informação) estimou que a quantidade de dispositivos enviados por todo o mundo entre 2014 e 2017, dentre estes computadores tradicionais (*desktops* e *notebooks*), *ultramobiles premium* (inclui MacBook *Air* e produtos da Microsoft com Windows 8 e Intel x86), *ultramobiles* (*tablets* e *clamshells*) e celulares aumentaria de 2,4 para 2,6 bilhões de unidades, sendo que destes, aproximadamente 2,1 bilhões são referentes a celulares. O Gráfico 3 apresenta os resultados obtidos.

Gráfico 3 - Estimativa da quantidade de dispositivos enviados pelo mundo (2014-2017)



Fonte: Adaptado de Gartner[41]

[40] Disponível em: <http://www.cisco.com/c/dam/assets/sol/sp/vni/forecast_highlights_mobile/index.html>. Acesso em jan. 2016.
[41] Disponível em: <http://www.gartner.com/newsroom/id/3088221>. Acesso em jan. 2016.

Com base nisso, desenvolver uma aplicação *web* na atualidade que seja adequada somente para uso em dispositivos com resolução maior é um erro visto o tamanho da fatia de mercado ocupada pelos celulares (GRÁFICO 3). Apesar das restrições de interface existente em meio aos variados tipos de dispositivos, existem muitos recursos que podem ser utilizados para a adaptação da mesma em diferentes tamanhos de telas. Um destes recursos, objeto desta seção é o *Design Web* Responsivo.

Design Web Responsivo se trata de um conjunto de técnicas (HTML, CSS e JavaScript) utilizadas para melhorar a experiência do usuário independente do dispositivo que ele esteja utilizando, permitindo por exemplo, que um *site* seja programado de forma que os elementos que o compõem se adaptem automaticamente ao tamanho de tela do dispositivo no qual está sendo visualizado. Muitos *frameworks* para desenvolvimento *web*, como o *Bootstrap* por exemplo, disponibilizam estilos CSS e *scripts* JavaScript para a aplicação do *design* responsivo. Em suma, um *design* responsivo deve incluir (TEIXEIRA, 2012):

- Adaptar o *layout* da página de acordo com a resolução em que está sendo visualizada.
- Redimensionar as imagens automaticamente para que caibam na tela e para que não sobrecarreguem a transferência de dados em um celular, por exemplo.
- Simplificar elementos da tela para dispositivos móveis, onde o usuário normalmente tem menos tempo e menos atenção durante a navegação.
- Ocultar elementos desnecessários nos dispositivos menores.
- Adaptar tamanho de botões e links para interfaces *touch* onde o ponteiro do mouse é substituído pelo dedo do usuário.
- Utilizar de forma inteligente recursos *mobile* como geolocalização e mudança na orientação do aparelho (horizontal ou vertical).

O *design* responsivo é uma das preocupações no desenvolvimento do sistema Pipou, visto que os dispositivos móveis, como *smartphones* e *tablets* vem sendo utilizados cada vez mais. Portanto, toda a aplicação vem sendo desenvolvida para ser adaptável aos diversos tipos de dispositivos, permitindo uma navegação sem perda de qualidade e usabilidade.

3 MATERIAIS E MÉTODOS

Esta seção detalha o desenvolvimento deste trabalho, como se deu o estudo das tecnologias aqui empregadas, como fez-se o levantamento de requisitos, esboços de telas elaborados, modelagem da aplicação, o ambiente de desenvolvimento e as ferramentas utilizadas e como implementaram-se as funcionalidades com base no *framework* Laravel, nos diversos *plug-ins* utilizados e na segurança dos dados.

A implementação desta versão inicial do sistema de informação Pipou é resultado de dois Trabalhos de Conclusão de Curso que ocorreram de maneira concomitante, sendo um deles o presente trabalho, e o outro o desenvolvimento de um motor de busca para documentos PDF que será usado na gestão documental do SGP, ambos sob a orientação do mesmo docente. Quando houver referência neste texto para o termo ‘equipe do projeto’, ‘equipe de desenvolvimento’ ou termos afins entende-se um grupo formado pelos dois pesquisadores mencionados e seu orientador.

3.1 Estudo das tecnologias

Inicialmente, estudaram-se as tecnologias para interface e interação com o usuário, como HTML, CSS e jQuery, em suas versões mais atuais. Realizou-se tal estudo por meio de encontros presenciais entre os pesquisadores dos dois trabalhos, onde um destes possuía experiência prévia com o desenvolvimento *web* e pôde ajudar com as orientações iniciais sobre estas tecnologias.

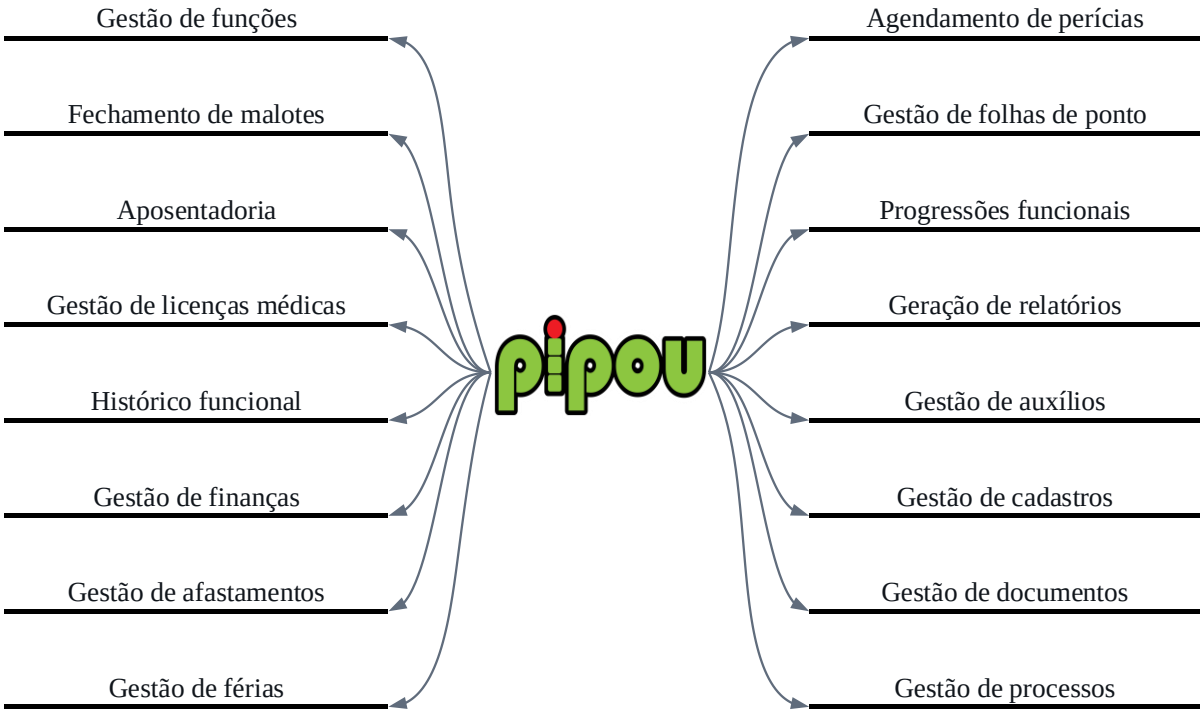
Como próximo passo, pesquisou-se sobre qual *framework* PHP utilizar para desenvolvimento *back-end*, uma vez que a linguagem escolhida para codificação do lado servidor foi o PHP. Em meio as vantagens e características do *framework* PHP Laravel (ver seções 2.4.7 e 2.4.8), optou-se por utilizá-lo para implementação do lado servidor. Deu-se início então ao estudo sobre o *framework* Laravel. Através da documentação oficial, livros, *sites* na internet e vídeo aulas, pequenos exemplos foram sendo desenvolvidos a fim de adquirir um conhecimento mais aprofundado sobre o funcionamento da ferramenta.

Por fim, com uma base mais sólida de conhecimento de HTML5, CSS3, jQuery e Laravel, fez-se o levantamento de requisitos da aplicação, este que será abordado com mais detalhes na seção 3.2.

3.2 Levantamento dos requisitos

Realizou-se o levantamento de requisitos da aplicação sob a forma de reuniões presenciais, onde o Coordenador do SGP enumerou os diversos processos executados pelo setor e os apresentou para os membros da equipe de desenvolvimento. Logo na primeira reunião solicitou-se que a Coordenação do SGP realizasse um levantamento contendo o máximo de tarefas exercidas pelo SGP, bem como possíveis funcionalidades do sistema que auxiliariam na automatização destas tarefas. Apresentou-se tal levantamento em reunião para que os requisitos do sistema fossem levantados e refinados em conjunto com a equipe de desenvolvedores. Pela descrição dos processos executados pelo setor apresentada na seção 2.1.1 chegou-se à conclusão que as funcionalidades desejáveis para compor o sistema Pipou seriam as dispostas no diagrama ilustrado pela Figura 21.

Figura 21 - Funcionalidades levantadas para o sistema Pipou



Fonte: Elaborada pelo autor

De posse das informações necessárias para tomada de decisão, a equipe verificou que o esforço a ser despreendido no desenvolvimento de um sistema de informações completo capaz de atender a todos os processos apresentados pela Coordenação do SGP, a complexidade do sistema, a disponibilidade de tempo para desenvolvimento e o tamanho da equipe impossibilitariam a implementação de todos os recursos necessários para contemplar completamente os processos do SGP. Isto posto, optou-se em reduzir o escopo do projeto neste Trabalho de Conclusão de Curso (TCC) a fim de cobrir em um primeiro momento apenas as funcionalidades mais relevantes para o setor: Gestão de Cadastros e Gestão de Documentos. Apesar da limitação do escopo, realizaram-se algumas atividades considerando o sistema final, como é o caso do levantamento de requisitos, os *mockups*[42] de interface e a modelagem do banco de dados.

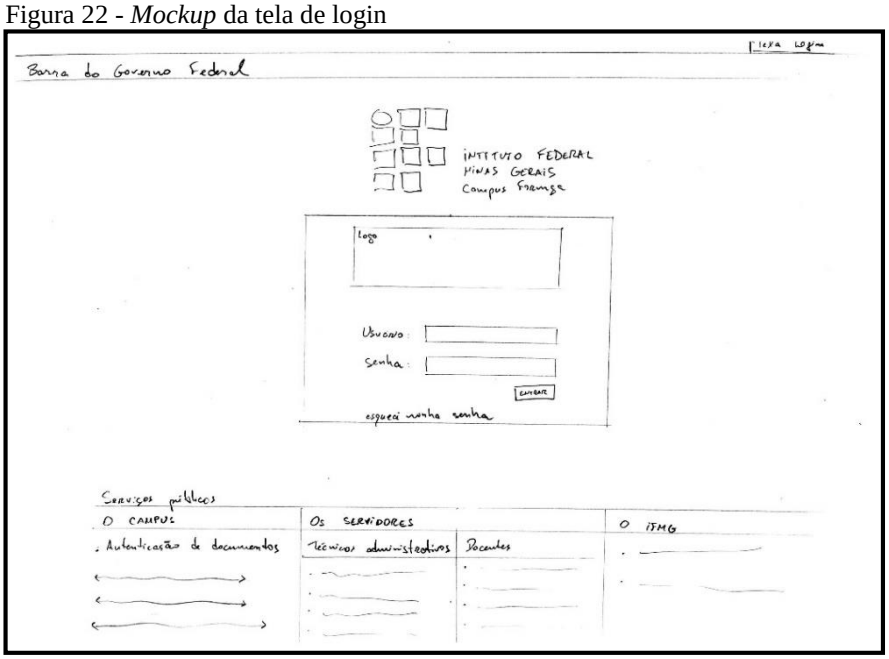
3.3 *Mockups* elaborados

Após o levantamento dos requisitos, como parte da concepção do sistema foram elaborados diversos *mockups* das interfaces do sistema com objetivo de colaborar para a formação de ideias e prover um desenvolvimento mais sólido. As Figuras 22, 23 e 24 ilustram alguns esboços de telas elaborados pela equipe.

A Figura 22 ilustra um exemplo de tela de *login* para o sistema, a qual provê os campos para preenchimento das credenciais do usuário, um *link* para recuperação de senha e *links* úteis para os serviços públicos no rodapé da página.

[42] Representação elaborada em tamanho real do que pode ser o produto final, um esboço de *layout*.

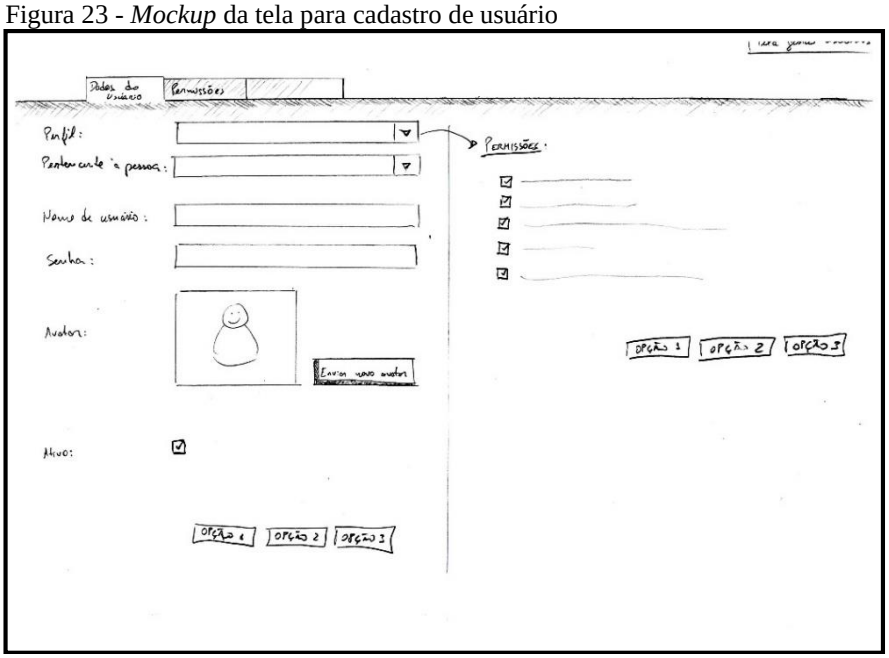
Figura 22 - *Mockup* da tela de login



Fonte: Elaborada pelo autor

A Figura 23 demonstra um *mockup* da tela onde o usuário administrativo cadastrará um servidor através do preenchimento dos campos providos entre as abas. Nesta tela, o usuário administrativo também fornece permissões de acesso ao servidor cadastrado.

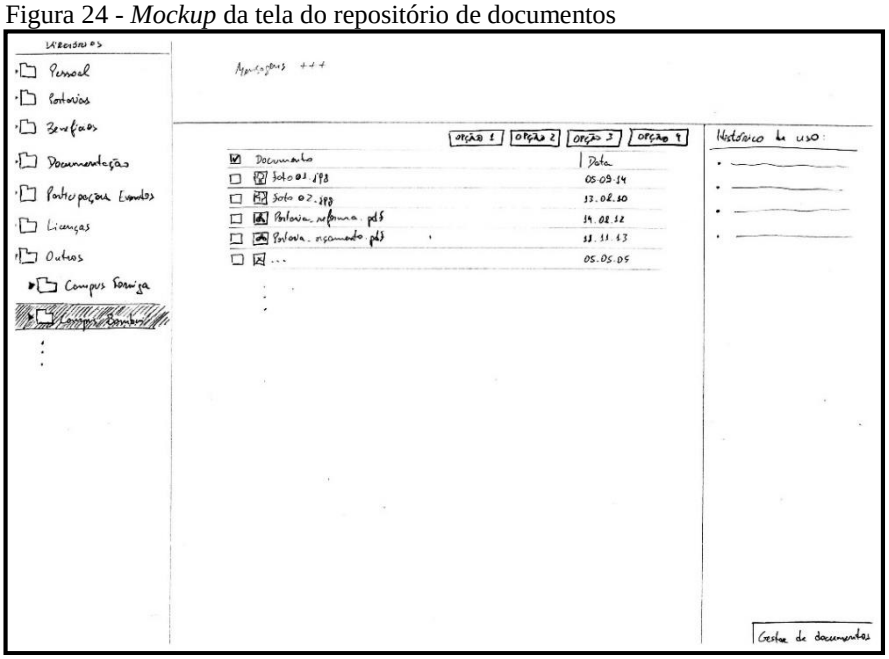
Figura 23 - *Mockup* da tela para cadastro de usuário



Fonte: Elaborada pelo autor

A Figura 24 é um exemplo do repositório de documentos utilizado pelo usuário administrativo, que exibe os documentos armazenados na pasta selecionada, provendo opções para executar ações, como adiciona e excluir, por exemplo. Além disso, provê histórico de uso contendo as últimas ações executadas no repositório.

Figura 24 - *Mockup* da tela do repositório de documentos



Fonte: Elaborada pelo autor

3.4 Ambiente de desenvolvimento

Realizou-se a implementação do sistema de informação Pipou em computadores pessoais pertencentes aos membros da equipe de desenvolvedores. Tais equipamentos possuem as seguintes configurações:

Microcomputador:

- Processador Intel® Core™ i7-4771 CPU @ 3.5GHz.
- 8GB de memória RAM DDR3 1600MHz.
- Disco rígido de 1TB 7200RPM.
- Sistema operacional Windows 8.1 *Professional* - 64 bits.

Notebook:

- Processador Intel® Core™ i3-2310M CPU @ 2.1GHz.
- 8GB de memória RAM DDR3 1333MHz.
- Disco rígido híbrido de 500GB com 8GB SSD 5400RPM.
- Sistema operacional Windows 8.1 *Professional* - 64 bits.

Notebook:

- Processador Intel® Core™ i7-4500U CPU @ 1.8 GHz - 2.7 GHz
- 8 GB da memória RAM DDR3 1600MHz.
- Disco rígido de 1TB 7200RPM.
- Sistema operacional Windows 8.1 *Professional Single Language* - 64 bits.

3.4.1 Ferramentas utilizadas

Fizeram-se necessárias diversas ferramentas para o desenvolvimento da aplicação, algumas citadas na seção 2, sendo estas:

- **TortoiseSVN v1.9.2** (https://tortoisesvn.net/): como o desenvolvimento inicial da aplicação se deu em conjunto, fez-se necessário um *software* para controle de versão, a fim de compartilhar as mudanças aplicadas por cada um dos membros da equipe, mantendo os arquivos atualizados.
- **WampServer v2.5** (http://www.wampserver.com/en/): por se tratar de um sistema *web*, fez-se necessário seu uso de modo a permitir o desenvolvimento da aplicação localmente no ambiente Windows, sem a necessidade de hospedagem. Nele estão incluídas as instalações do servidor Apache v2.4.9, do banco de dados MySQL v5.6.17 e do PHP v5.5.12.
- **Sublime Text 3** (http://www.sublimetext.com/3): editor de texto utilizado na codificação a fim de facilitar sua escrita, aumentado a produtividade.
- ***Template* AdminLTE v2.0** (https://almsaeedstudio.com/preview): usado como modelo de interface para a aplicação. Dessa forma, poupou-se muito tempo gasto na implementação de uma interface inicial.
- **MySQL Workbench v6.3 CE** (https://dev.mysql.com/downloads/workbench/): interface gráfica utilizada para manipular o banco de dados da aplicação.
- **Google Chrome** (https://www.google.com.br/chrome/browser/desktop/) **e Mozilla Firefox** (https://www.mozilla.org/pt-BR/firefox/new/): navegadores utilizados para realizar os testes com a aplicação, verificando seu comportamento e apresentação, sendo o primeiro em maior parte.

- **Laravel v4.2.17**.
- **jQuery v2.1.3 e jQuery UI[43] v1.11.4**.
- ***Bootstrap* v3.3.4**.

Com exceção da ferramenta Sublime Text 3, todas as outras supracitadas são gratuitas.

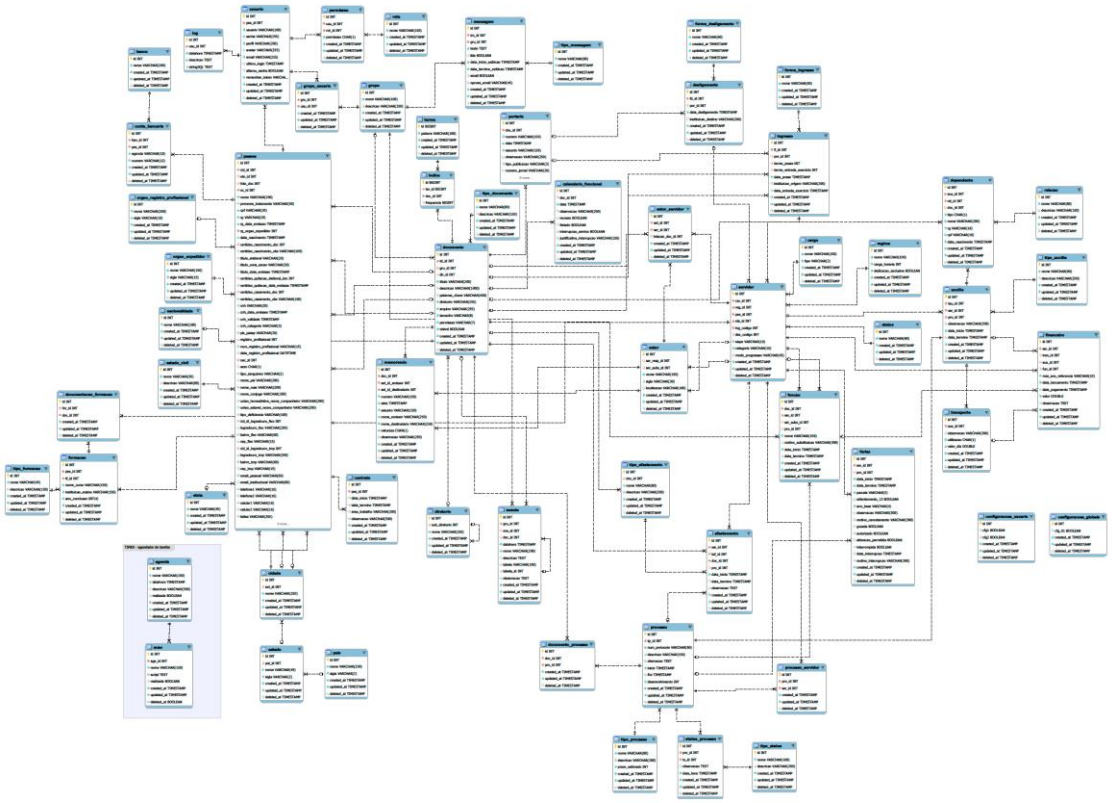
3.5 Modelagem da aplicação

Modelou-se a aplicação com base nos requisitos levantados. Executou-se tal etapa pensando na persistência dos dados, isto é, modelou-se um Diagrama de Entidade Relacional (DER) contendo todas as tabelas e relacionamentos necessários para armazenar os dados da aplicação. Para a criação do DER, utilizou-se o *software* MySQL Workbench v6.3 CE. Cabe ressaltar que embora o escopo da implementação deste trabalho seja reduzido em virtude da complexidade do *software*, disponibilidade de tempo e tamanho de equipe, fez-se a modelagem do banco de dados para o sistema completo em sua versão final.

A medida que os requisitos do sistema evoluíram, fizeram-se alterações na modelagem. Ao todo, 62 tabelas compõem o modelo do banco de dados da aplicação completa, sendo que 29 destas são contempladas neste trabalho. A Figura 25 mostra o DER modelado para o sistema Pipou.

[43] Biblioteca JavaScript com um conjunto de interações de interface de usuário, efeitos, *widgets* e temas construídos em cima da biblioteca jQuery. Disponível em: <http://jqueryui.com/download/>. Acesso em jan. 2016.

Figura 25 - Modelagem do DER da aplicação



Fonte: Elaborada pelo autor

O Quadro 32 apresenta as tabelas utilizadas no desenvolvimento deste trabalho e uma breve descrição de sua função e local de utilização no sistema.

Quadro 32 - Tabelas utilizadas no desenvolvimento do trabalho (continua)

Nome	**Descrição**
banco	Usada para armazenar os bancos financeiros cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de bancos e pessoas.
cargo	Usada para armazenar os cargos cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de cargos e servidores.
cidade	Usada para armazenar as cidades cadastradas na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de cidades e pessoas.
conta_bancaria	Usada para armazenar as contas bancárias de uma pessoa na aplicação. Utilizada no cadastro de pessoas.

Quadro 32 - Tabelas utilizadas no desenvolvimento do trabalho (continuação)

Nome	**Descrição**
documentacao_formacao	Usada para armazenar informações dos documentos referentes às documentações comprobatórias das formações acadêmicas da pessoa cadastrada na aplicação. Utilizada no cadastro de pessoas.
documento	Usada para armazenar as informações dos documentos enviados para o servidor. Utilizada nos cadastros de memorandos, ofícios, portarias, pessoas, servidores e no repositório digital de documentos.
estado	Usada para armazenar os estados cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de estados e cidades.
estado_civil	Usada para armazenar os estados civis. Previamente semeada com o recurso de *seeding* do Laravel. A aplicação não dispõe do cadastro de estados civis. Utilizada no cadastro de pessoas.
etnia	Usada para armazenar as etnias cadastradas na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de etnias e pessoas.
forma_desligamento	Usada para armazenar as formas de desligamento de um servidor cadastradas na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada no cadastro de servidores.
forma_ingresso	Usada para armazenar as formas de ingresso de um servidor cadastradas na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada no cadastro de servidores.
formacao	Usada para armazenar as formações acadêmicas de uma pessoa cadastrada na aplicação. Utilizada no cadastro de pessoas.
log	Usada para armazenar informações de *log* geradas pelo servidor. Utilizada em todas as telas do sistema.
memorando	Usada para armazenar os memorandos cadastrados na aplicação. Utilizada no cadastro de memorandos.
nacionalidade	Usada para armazenar as nacionalidades cadastradas na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de nacionalidades e pessoas.

Quadro 32 - Tabelas utilizadas no desenvolvimento do trabalho (continuação)

Nome	**Descrição**
oficio	Usada para armazenar os ofícios cadastrados na aplicação. Utilizada no cadastro de ofícios.
orgao_expedidor	Usada para armazenar os órgãos expedidores de RG cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada no cadastro de pessoas.
orgao_profissional	Usada para armazenar os órgãos profissionais (ex: CREA) cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada no cadastro de pessoas.
pais	Usada para armazenar os países cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de países e estados.
pasta	Usada para armazenar as pastas do repositório digital de documentos.
permissao	Usada para armazenar as permissões de acesso dos usuários para cada tela do sistema, baseado na tabela de rotas. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada em todas as telas do sistema.
pessoa	Usada para armazenar as pessoas cadastradas na aplicação. Utilizada nos cadastros de pessoas e servidores.
portaria	Usada para armazenar as portarias cadastradas na aplicação. Utilizada no cadastro de portarias.
regime	Usada para armazenar os regimes de trabalho de um servidor cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de regimes e servidores.
rota	Usada para armazenar as rotas de cada tela do sistema. Previamente semeada com o recurso de *seeding* do Laravel.
setor	Usada para armazenar os setores de lotação cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de setores, memorandos e ofícios.
status	Usada para armazenar os status que um servidor cadastrado pode assumir (ex: Ativo). Previamente semeada com o recurso de *seeding* do Laravel. Utilizada nos cadastros de status e servidores.

Quadro 32 - Tabelas utilizadas no desenvolvimento do trabalho (conclusão)

Nome	Descrição
tipo_formacao	Usada para armazenar os tipos de formações (ex: Graduação). Previamente semeada com o recurso de *seeding* do Laravel. O sistema não dispõe do cadastro de tipos de formações.
usuario	Usada para armazenar os usuários cadastrados na aplicação. Previamente semeada com o recurso de *seeding* do Laravel. O sistema ainda não dispõe do cadastro de usuários.

Fonte: Elaborado pelo autor

3.6 Implementação

A implementação do sistema de informação Pipou se deu em conjunto pelos dois desenvolvedores que formam a equipe de desenvolvimento, como foco inicial no módulo de Gestão de Cadastros. Implementou-se uma versão básica de cadastro primeiro, sendo posteriormente refinada e replicada aos demais. Em seguida desenvolveu-se um *script* genérico contendo funcionalidades para *upload* e *download* de documentos que é aplicado em alguns cadastros específicos e no repositório digital de documentos, implementado posteriormente.

No desenvolvimento *front-end* fez-se o uso dos recursos do *template* AdminLTE, *Bootstrap*, jQuery e CSS3. Para a implementação do *back-end* da aplicação utilizaram-se os recursos do Laravel. Atenta-se ao fato de que todos os recursos aplicados tanto no lado cliente como no servidor são inteiramente de código aberto, sendo customizados conforme a necessidade para atender às funcionalidades implementadas no sistema.

3.6.1 Desenvolvimento front-end

No lado cliente da aplicação, cada interface possui sua estilização e interação. No caso dos cadastros, existem vários estilos e *scripts* comuns a todos, e por isso foram criados dois arquivos padrões separados (um *.css* para os estilos e outro *.js* para os *scripts* JavaScript). Os estilos e interações específicos de cada tela estão contidos em arquivos *.css* e *.js* separados. Para

a parte de *upload*/*download* de documentos, criou-se um arquivo *.js* em separado contendo todas as funções necessárias para a realização destes. A funcionalidade de *upload*/*download* de documentos é utilizada pelos Cadastros de Documentos (ofícios, memorandos e portarias), Pessoas (certidões de nascimento, casamento e de quitação eleitoral, comprovantes de formação acadêmica) e pelo repositório digital de documentos para anexo e *download,* respectivamente.

Os *scripts* jQuery implementados manipulam todas as ações realizadas pelo usuário, como clique em botões, seleção de opções em combos, troca de abas, expansão de *accordions*[44], adição e remoção de campos entre outros. No caso dos cadastros, os arquivos *.js* também contém funções responsáveis pelo gerenciamento dos cadastros, enviando requisições via AJAX para adição, remoção, visualização e edição de registros. Os *scripts* para manipular o repositório de documentos lidam com a criação visual da estrutura de pastas, incluindo ações como adição, remoção e renomeação das mesmas, e o gerenciamento de documentos dentro de cada uma, imitando um repositório físico baseado nos diretórios de um computador, onde dentro de uma pasta podem ter vários documentos e várias pastas.

A fim de proporcionar mais funcionalidades ao sistema em menos tempo, melhorando sua qualidade total, além dos *plug-ins* disponibilizados pelo *Bootstrap* e pelo *template* AdminLTE, utilizaram-se vários outros no *front-end* da aplicação, permitindo uma interatividade maior com o usuário. O Quadro 33 lista todos os *plug-ins* utilizados até o momento na aplicação, bem como dão uma breve descrição de suas funções e o *link* para *download*/instalação dos mesmos.

Quadro 33 - *Plug-ins* utilizados no desenvolvimento *front-end* (continua)

Plug-in*	**Descrição**
Bootbox	Pequena biblioteca JavaScript que permite a criação de janelas modais programáticas usando *Bootstrap* sem ter que se preocupar sobre como criar, gerir ou remover qualquer um dos elementos DOM necessários ou manipuladores de eventos JavaScript. (http://bootboxjs.com/)
DataTables	*Plug-in* jQuery que adiciona controles avançados de interação em qualquer tabela HTML. Permite a criação de tabelas de dados com processamento *server-side*, além de prover funcionalidades de ordenação, pesquisa etc. (https://www.datatables.net/)

[44] Elementos usados para expandir e recolher o conteúdo que é dividido em seções lógicas. Leva esse nome por lembrar um acordeão ao expandir e recolher. Muito utilizado para exibir conteúdos em espaços limitados.

Quadro 33 - *Plug-ins* utilizados no desenvolvimento *front-end* (conclusão)

Plug-in*	**Descrição**
Dropzone	Realiza o *upload* de arquivos no modo *drag and drop*. Também provê estilos para pré-visualização de imagens. É leve e, embora possa ser utilizado em conjunto com jQuery, não depende deste assim com de nenhuma outra biblioteca. (http://www.dropzonejs.com/)
jQuery *File Download*	*Plug-in* jQuery que permite a realização de *downloads* via AJAX, o que não é normalmente possível usando a *web*. (https://github.com/johnculviner/jquery.fileDownload)
jQuery *Input Mask*	*Plug-in* jQuery para criação de máscaras nos campos dos formulários, como CPF, datas, telefones, valores numéricos etc. (https://github.com/RobinHerbots/jquery.inputmask)
jQuery *Validation*	*Plug-in* jQuery que faz a validação de formulário do lado do cliente. Baseia-se nos atributos inseridos nas *tags* HTML e provê um conjunto vasto de métodos de validação para campos obrigatórios, e-mail, url, datas, cartão de crédito entre outros. (http://jqueryvalidation.org/)
jsTree	*Plug-in* jQuery que permite a construção de *treeviews* interativos. Suporta fontes de dados HTML, JSON e AJAX. Útil para criar estruturas de diretórios. (https://www.jstree.com/)
jQuery *Watch*	*Plug-in* jQuery que permite monitorar alterações de estilos CSS, atributos ou propriedades de qualquer elemento DOM e disparar um *callback* em resposta a qualquer mudança no elemento monitorado. (https://github.com/RickStrahl/jquery-watch)
PlusAsTab	*Plug-in* jQuery que permite uma tecla emular o funcionamento da tecla Tab. No caso específico deste trabalho, o ENTER. (https://github.com/joelpurra/plusastab)

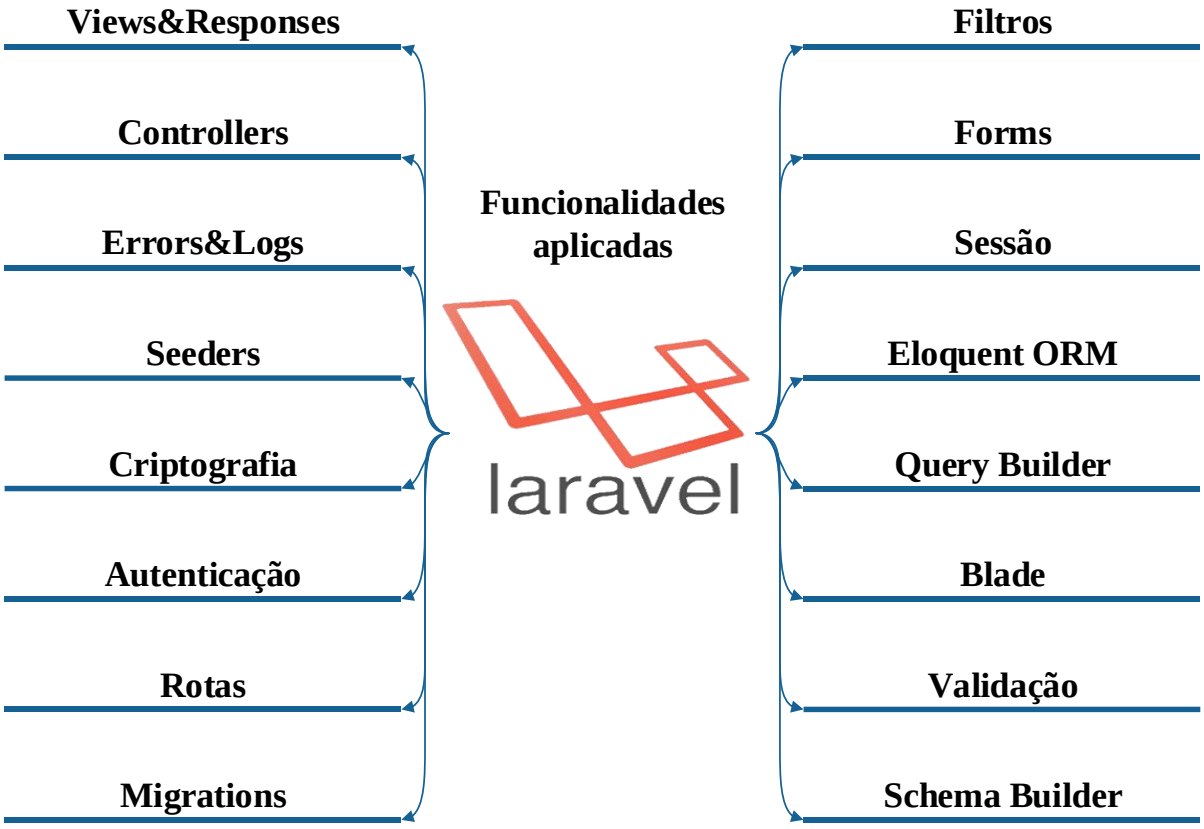
Fonte: Elaborado pelo autor

* Todos são gratuitos e de código aberto

3.6.2 Desenvolvimento back-end

O desenvolvimento da parte *back-end* do sistema de informação Pipou utilizou boa parte dos recursos fornecidos pelo *framework* Laravel. A Figura 26 apresenta um diagrama com os recursos que foram usados em algum momento na implementação do sistema.

Figura 26 - Recursos do Laravel utilizados na aplicação



Fonte: Elaborada pelo autor

O Quadro 34 descreve brevemente o contexto de uso destes recursos na aplicação.

Quadro 34 - Contexto de uso dos recursos do Laravel na aplicação (continua)

Recurso	**Contexto de aplicação**
Views&Responses	Utilizado para o envio de páginas HTML e respostas diversas ao usuário, como JSON, texto plano, HTML, *download* etc.
Controllers	Cada tela do sistema possui um *controller* para tratar suas requisições.
Errors&Logs	Utilizado nos *controllers* para emissão de erros e depuração da aplicação.

Quadro 34 - Contexto de uso dos recursos do Laravel na aplicação (continuação)

Recurso	**Contexto de aplicação**
Seeders	Utilizado para semear o banco de dados com dados de: bancos, cargos, cidades, estados, estados civis, etnias, formas de desligamento e ingresso, nacionalidades, países, permissões de acesso a rotas, rotas acessíveis, setores, sexo, status do servidor, tipos de cargos, tipos de formações acadêmicas, usuários.
Criptografia	Utilizado para criptografar e descriptografar dados sigilosos retornados pelo servidor, como o id de um registro na tabela do banco de dados ou a senha do usuário por exemplo.
Autenticação	Utilizado para autenticar o usuário na aplicação, criando sua sessão e mantendo-o *logado*.
Rotas	Utilizado para capturar as requisições feitas ao servidor, tratando ou encaminhando-as para um *controller*.
Migrations	Utilizado para criar a base de dados e aplicar alterações na mesma.
Filtros	Utilizado para filtrar as requisições feitas ao servidor, impedindo o usuário de acessar determinadas rotas sem estar *logado* e protegendo a aplicação de ataques *Cross-site Request Forgery* (CSRF, Falsificação de solicitação entre *sites*)[45].
Forms	Sintaxe *blade* utilizada para criar um elemento <form>, atribuindo-lhe um *token* criptografado referente àquela sessão, protegendo a aplicação de ataques CSRF.
Sessão	Utilizado para recuperar atributos da sessão e para armazenagem desta no banco de dados através do *driver database* do Laravel.
Eloquent ORM	Utilizado para mapear toda a base de dados através de classes, facilitando a realização de consultas e operações CRUD.
Query Builder	Utilizado em alguns casos onde as consultas foram muito específicas ou complexas, sendo difícil realizá-las com o *Eloquent*.
Blade	Utilizado para diminuir a quantidade de código nas páginas HTML, deixando-as mais legíveis.

[45] Tipo de ataque informático malicioso a um *website* no qual comandos não autorizados são transmitidos através de um utilizador em quem o *website* confia. Mais em: <https://pt.wikipedia.org/wiki/Cross-site_request_forgery>. Acesso em jan. 2016.

Quadro 34 - Contexto de uso dos recursos do Laravel na aplicação (conclusão)

Recurso	**Contexto de aplicação**
Validação	Utilizado em todas as telas implementadas para validar os dados do lado do servidor, evitando possíveis inconsistências.
Schema Builder	Utilizado nas *migrations* para criar a base de dados.

Fonte: Elaborado pelo autor

Do lado do servidor, cada interface da aplicação possui um arquivo *.blade.php* contendo todo o conteúdo estruturado em HTML e PHP. Como existem seções fixas comuns a todas as telas do sistema, como o menu principal, a barra lateral, o cabeçalho e o rodapé, fez-se o uso dos *templates Blade* do Laravel, adicionando somente o conteúdo específico de cada página nos arquivos de visões, melhorando a legibilidade e diminuindo a quantidade de código repetido. No caso dos cadastros, muitos arquivos de estilo e *scripts* são comuns a todos, onde criaram-se dois arquivos *.blade.php* contendo todos os comandos para adição dos mesmos no documento de visão. Para cada tabela modelada no banco de dados, precisou-se criar uma classe PHP específica para correta utilização do ORM. Estas classes são o *Model* do modelo MVC. Da mesma forma, para tratar as requisições de cada página, implementou-se uma classe PHP para cada tela, a fim de realizar a função de *Controller*. Além disso, duas classes PHP foram criadas para prover métodos úteis e de segurança à aplicação. A primeira contém métodos como criação de *breadcrumbs*, armazenamento de *logs*, montagem de vetores utilizados em *combobox*, montagem de menu de acordo com as permissões do usuário etc. A segunda provê métodos para segurança do sistema, como alimentação e verificação de permissões do usuário na sessão e criptografia e descriptografia de dados.

As tabelas do banco de dados foram criadas por meio do recurso de *migrations* disponibilizado pelo Laravel, onde em sua maioria foram alimentadas com *seeders*, também fornecidos pelo *framework*. Atualmente, somente dois usuários são alimentados no banco para testes da aplicação, no entanto, utiliza-se os métodos fornecidos pelo Laravel para criptografar a senha antes de salvá-la e descriptografá-la ao verificar o *login* do usuário.

Como critérios de segurança, implementou-se um grupo de rotas nas quais todas elas passam pelo filtro de autenticação do Laravel para acesso, isto é, o usuário precisa ter efetuado *login* para acessar qualquer rota que esteja neste grupo. Além disso, criou-se um mecanismo para permissão de acesso do usuário. Para cada rota da aplicação (cada tela acessada), o usuário tem um nível de permissão: **r** para leitura e **w** para leitura/escrita. Caso o usuário possua uma dessas permissões, o mesmo acessa a interface correspondente a rota permitida; do contrário a opção de acesso àquela página não será mostrada no menu lateral do usuário. Embora o cadastro

de controle de usuários ainda não tenha sido implementado, este mecanismo foi testado pelos desenvolvedores e será aplicado futuramente, permitindo que o usuário administrativo tenha total liberdade na configuração de permissões de um dado usuário.

Embora o Laravel possua diversos recursos e permita a aplicação destes de forma mais facilitada que a maioria dos *frameworks*, em alguns casos algumas implementações tornam-se trabalhosas, o que ocasionou no desenvolvimento de inúmeros pacotes de extensão para o Laravel, os quais adotaram-se alguns para o desenvolvimento *back-end*. O Quadro 35 lista todos os pacotes utilizados e descreve brevemente suas funções, disponibilizando um *link* para *download*/instalação dos mesmos.

Quadro 35 - Pacotes utilizados no desenvolvimento *back-end*

Pacote*	**Descrição**
Active for Laravel	Permite adicionar uma classe no item de menu com base na url atual. Útil quando se deseja manter selecionado ou destacado o item de menu clicado depois que uma nova página é carregada. (https://www.hieule.info/products/active-class-helper-laravel-4/)
Faker	Biblioteca PHP que gera dados fictícios automaticamente. Muito útil para alimentar bancos de dados. (https://packagist.org/packages/fzaninotto/faker)
Laravel Datatables	Usando recursos do Laravel, tais como ORM *Eloquent*, *Query Builder* ou *Collection*, manipula do lado do servidor as requisições feitas pelo *plug-in* DataTables via AJAX. (http://datatables.yajrabox.com/)

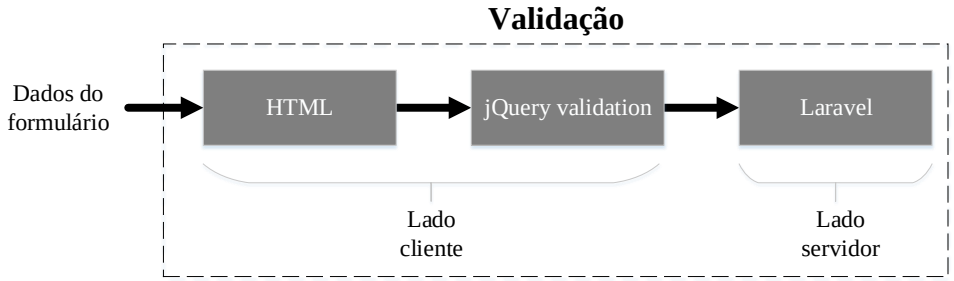
Fonte: Elaborado pelo autor
* Todos são gratuitos e de código aberto

3.6.3 Preocupações com a consistência dos dados

Por se tratar de uma aplicação que armazena dados sensíveis de servidores, como contas bancárias, CPF, RG etc., o sistema deve conter mecanismos para validar todas as informações dos cadastros antes de persisti-las no banco de dados, a fim de evitar inconsistência de dados importantes relacionados a vida profissional do servidor. Para isso, implementou-se uma validação em três camadas, de modo que os campos primeiramente são validados pelo próprio

HTML, através dos atributos inseridos nas *tags*. Em seguida, validados pelo HTML, os mesmos campos são validados via JavaScript pelo *plug-in* jQuery *Validation*. Por fim, para correta persistência, os dados enviados pelo formulário são validados do lado do servidor pelos recursos de validação fornecidos pelo *framework* Laravel. Somente após passar pelas três camadas de validação é que o registro é salvo no banco de dados. A Figura 27 ilustra a sequência de validação em três camadas.

Figura 27 - Validação em três camadas



Fonte: Elaborada pelo autor

4 O SISTEMA PIPOU

O nome “Pipou” é uma referência ao termo “*people*” em inglês que significa “pessoas”. Optou-se por este nome em meio ao contexto ao qual o sistema será inserido, a Gestão de Pessoas. O sistema conta com um módulo cadastral, onde se pode cadastrar registros básicos como cidades, pessoas, servidores, cargos etc., e com um repositório de digital de documentos, onde é possível gerenciar documentos digitais por meio de pastas virtuais, imitando os diretórios encontrados nos sistemas operacionais modernos dos computadores. O objetivo inicial desta versão aqui apresentada é a de permitir a alimentação dos dados através dos cadastros e gerir de forma simplificada a documentação gerada pelo setor. Assim, poder-se-á coletar *feedbacks* dados pela Coordenadoria do SGP do IFMG *campus* Formiga, que serão utilizados para aplicação de melhorias e funcionalidades futuras. Cabe ressaltar que algumas interfaces implementadas não são ainda funcionais, pois lidam com partes do sistema que ficaram de fora do escopo da implementação do presente Trabalho de Conclusão de Curso, mas estão disponíveis na camada de visão à espera de sua implementação em oportunidades futuras.

4.1 Tela de *login*

A tela de *login* (autenticação) é simples e contém os campos para inserção de usuário e senha. Possui uma barra superior com o logotipo da aplicação e um *link* para o “Fale conosco!” (ícone de envelope à direita do menu superior), funcionalidade ainda não implementada, mas que consistirá de um formulário para contato com os administradores do sistema ou com a Coordenadoria do SGP do *campus* Formiga. Conta também com uma botão “Esqueci minha senha!”, o qual alterna o formulário de *login* para a interface de recuperação de senha. Nela o usuário insere o e-mail de cadastro, para o qual será enviado a senha cadastrada. Esta funcionalidade permanece na lista de funcionalidades futuras pois ainda não foi implementada uma vez que não optou-se pelo cadastro de usuários como prioritário na definição do escopo deste Trabalho de Conclusão de Curso. A Figura 28 apresenta a interface de tela de *login*.

Figura 28 - Tela de *login* da aplicação

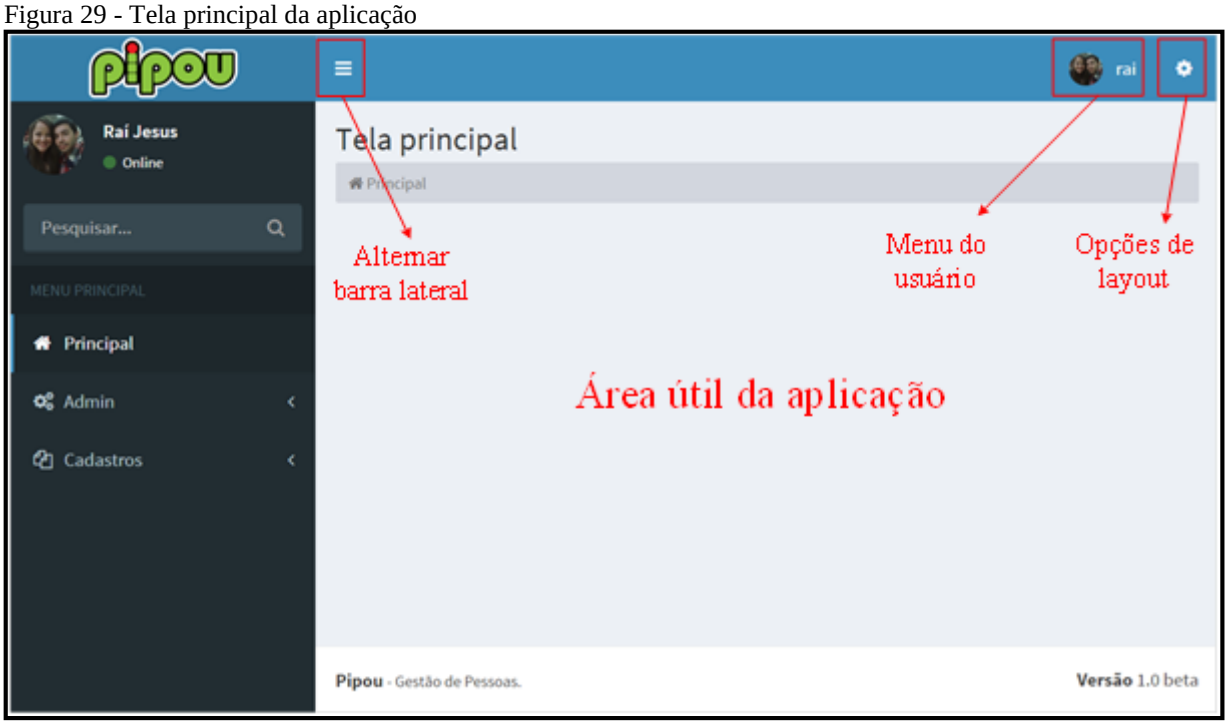
Fonte: Elaborada pelo autor

4.2 Tela principal

A tela principal do sistema é composta por um cabeçalho principal, uma barra lateral, a área útil da aplicação e um rodapé. O cabeçalho contém botões para alternância da barra lateral

(mostrar/ocultar), menu do usuário e opções de *layout*. Na barra lateral está o menu principal, com opções administrativas e de cadastros, mostradas de acordo com o nível de permissão do usuário. A área útil é a parte onde é apresentado o conteúdo da página, como formulários, tabelas, botões etc. O rodapé contém apenas informações básicas da aplicação, como nome e versão. A Figura 29 mostra a interface da tela principal, destacando alguns dos componentes.

Figura 29 - Tela principal da aplicação



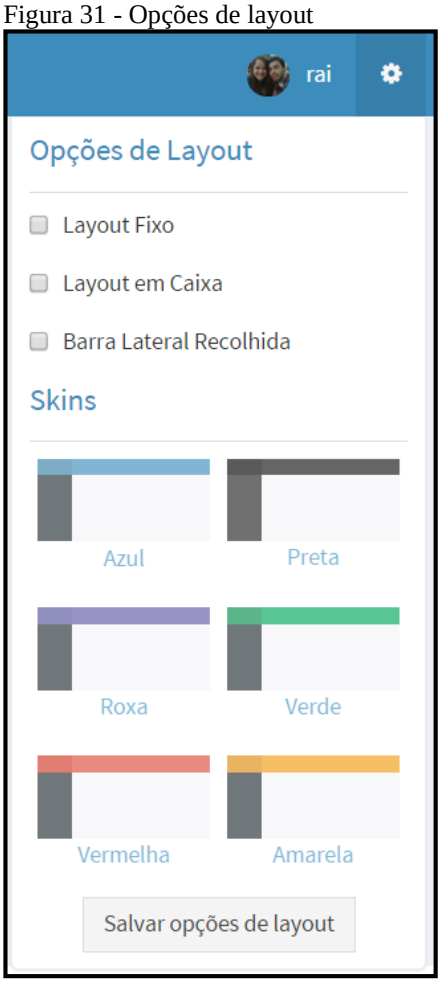
Fonte: Elaborada pelo autor

Os elementos “Menu do usuário” e “Opções de layout” destacados na Figura 28, são expandidos para baixo ao serem clicados. O menu do usuário mostra informações do usuário, como foto de perfil, data de cadastro, data e hora do último acesso, além de dois botões na parte inferior, um para exibir o perfil do usuário (ainda não implementado) e outro para sair da aplicação (*logout*). As opções de *layout* permitem alterar a interface do sistema, com opções para exibição de *layout* fixo, em caixa ou com a barra lateral recolhida, e para mudar as cores de alguns elementos. Também conta com um botão para salvar as opções definidas pelo usuário no banco de dados, carregando-as quando o mesmo efetuar *login*. Porém, como o sistema terá futuramente inúmeras configurações globais, decidiu-se não implementar o salvamento das opções de *layout* neste trabalho. As Figuras 30 e 31 apresentam os elementos menu do usuário e as opções de *layout* expandidos, respectivamente.

Figura 30 - Menu do usuário

Fonte: Elaborada pelo autor

Figura 31 - Opções de layout



Fonte: Elaborada pelo autor

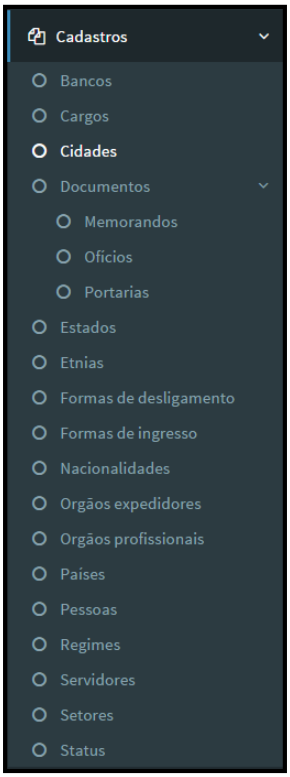
4.3 Módulo de cadastros

O módulo de cadastros conta atualmente com 19 cadastros. O sistema dá total liberdade para o usuário cadastrar a maioria dos dados que serão utilizados na aplicação. A página de cadastro contém duas abas, “Pesquisa de <nome_do_cadastro>” e “Gestão de <nome_do_cadastro>”. A primeira exibe em forma tabular os registros já cadastrados. O *plug-in jQuery DataTables* constrói a tabela, permitindo ao usuário aplicar ações como pesquisa, ordenação e seleção de registros. Esta mesma aba contém botões de ações para adição de novo registro, visualização e exclusão. A segunda contém os *inputs* necessários para adicionar, visualizar ou editar um registro. Esta última também contém botões de ações, os quais se alteram conforme o modo em que se encontra o cadastro. Há três modos disponíveis: Inserção, Visualização e Edição.

No modo Inserção e Edição, os campos estão habilitados para preenchimento e são exibidos os botões para cancelar a ação, limpar o formulário e salvar o registro. No modo Visualização, são exibidos os botões para adicionar um novo registro e editar o registro visualizado. Além disso, como medida de segurança e consistência dos dados toda a interação possível (digitação, *backspace* etc.) com os campos é desabilitada neste modo. Tirando os elementos que por padrão estão desabilitados ou somente leitura, alguns (como botões, *combobox*, *checkbox* etc.) também são, e caso o usuário tente forçar alterar suas propriedades, o *plug-in* jQuery Watch cuida de não permitir isso. No modo Visualização, somente os botões de anexo de documentos ficam habilitados caso o registro em visualização possua algum documento anexado referente àquele botão.

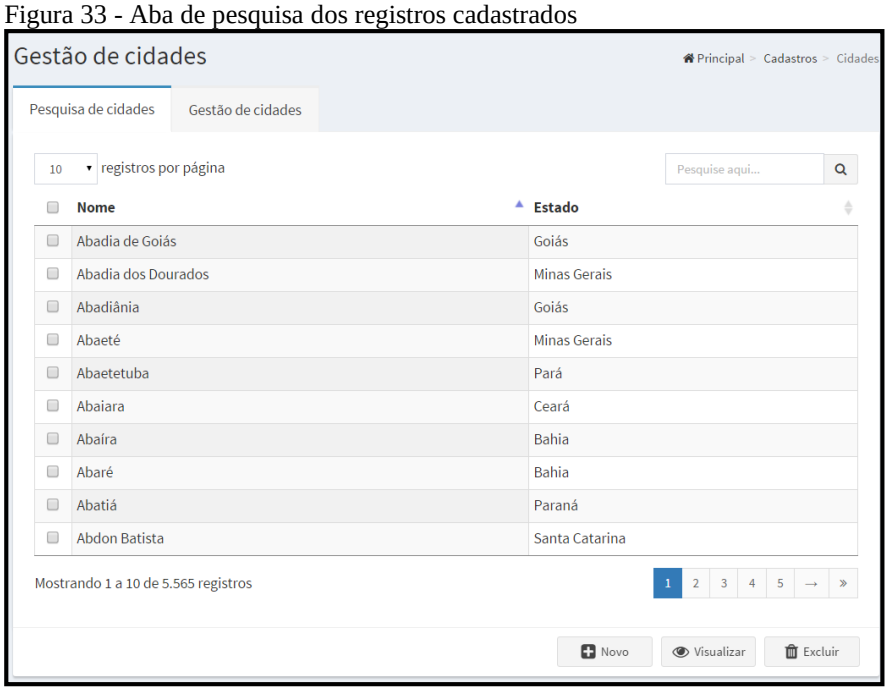
As Figuras 32, 33 e 34 mostram respectivamente o menu com as opções de cadastro e as abas de pesquisa e gestão de cadastro.

Figura 32 - Menu de cadastros



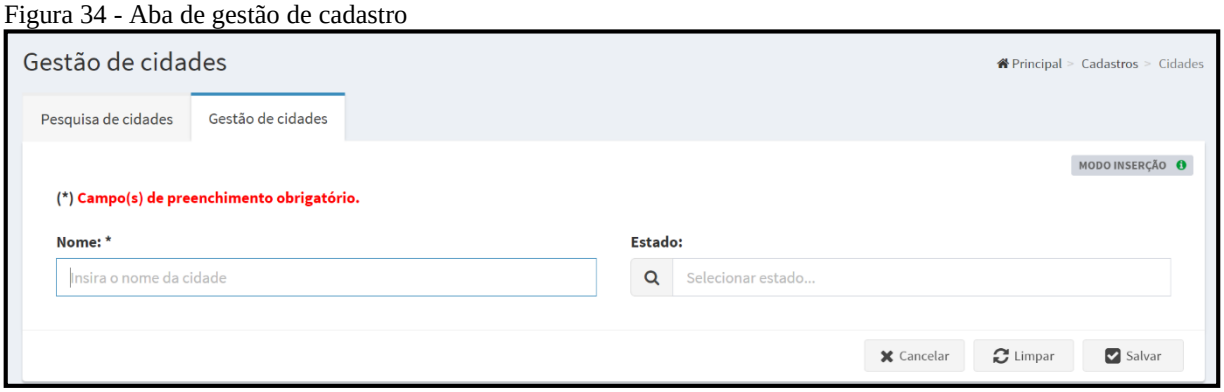
Fonte: Elaborada pelo autor

Figura 33 - Aba de pesquisa dos registros cadastrados



Fonte: Elaborada pelo autor

Figura 34 - Aba de gestão de cadastro

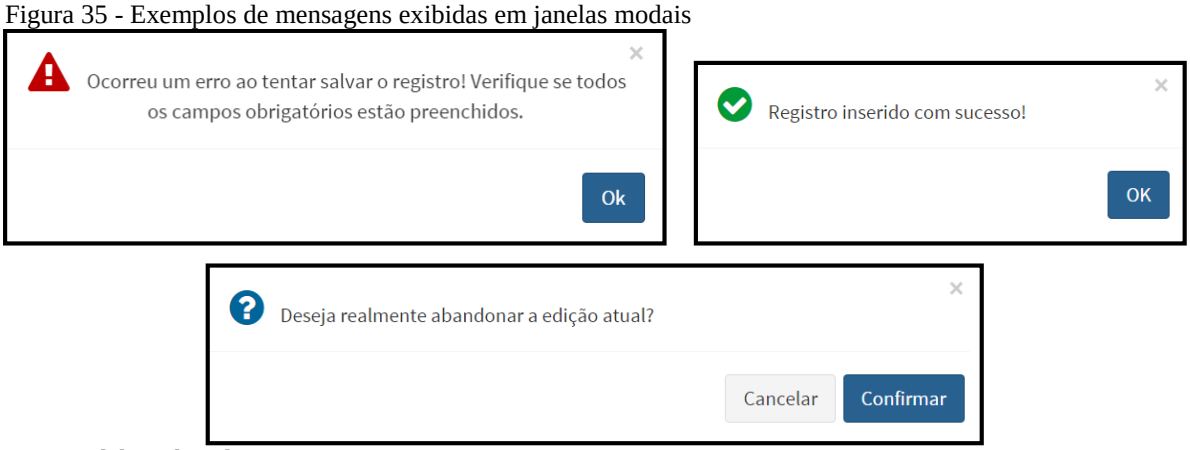


Fonte: Elaborada pelo autor

Todos os cadastros emitem mensagens em janelas modais para informar o usuário sobre resultado do processamento de determinada ação (FIGURA 35). Isso possibilita ao usuário mais controle da aplicação, além de torná-la mais segura, provendo uma usabilidade e experiência de uso mais agradável. O *plug-in* Bootbox permitiu a criação destas janelas modais de maneira facilitada. São exemplos de mensagens geradas pelo sistema:

- Tentar visualizar ou excluir nenhum registro ou mais do que um selecionado;
- Erros de validação nos campos do formulário;
- Confirmações de cancelamento de adição/edição do registro e *reset* de formulário;
- Sucesso ou erro no salvamento do registro;
- Limite de documentos anexados atingido entre outras.

Figura 35 - Exemplos de mensagens exibidas em janelas modais

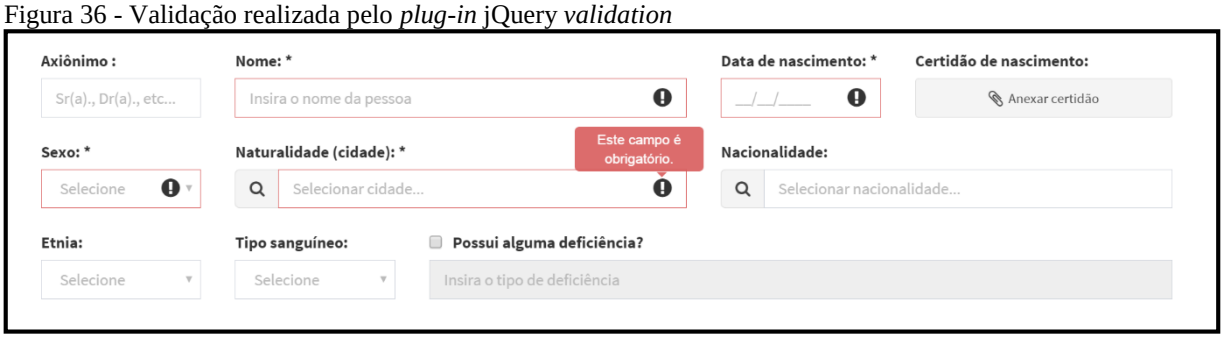


Fonte: Elaborada pelo autor

Os formulários são validados pelo *plug-in* jQuery *Validation* (ver seções 3.6.1 e 3.6.3). Caso exista algum erro, uma janela modal é exibida alertando o usuário, indicando os campos inválidos em vermelho com um ícone de exclamação em seu interior que exibe o erro ao passar o mouse sobre ele. A Figura 36 apresenta a validação do jQuery *Validation* operando,

destacando os campos que não passaram pelo crivo da validação. No caso de ocorrência de erro na validação do lado do servidor uma janela modal também é exibida, listando todos os erros de validação identificados no servidor.

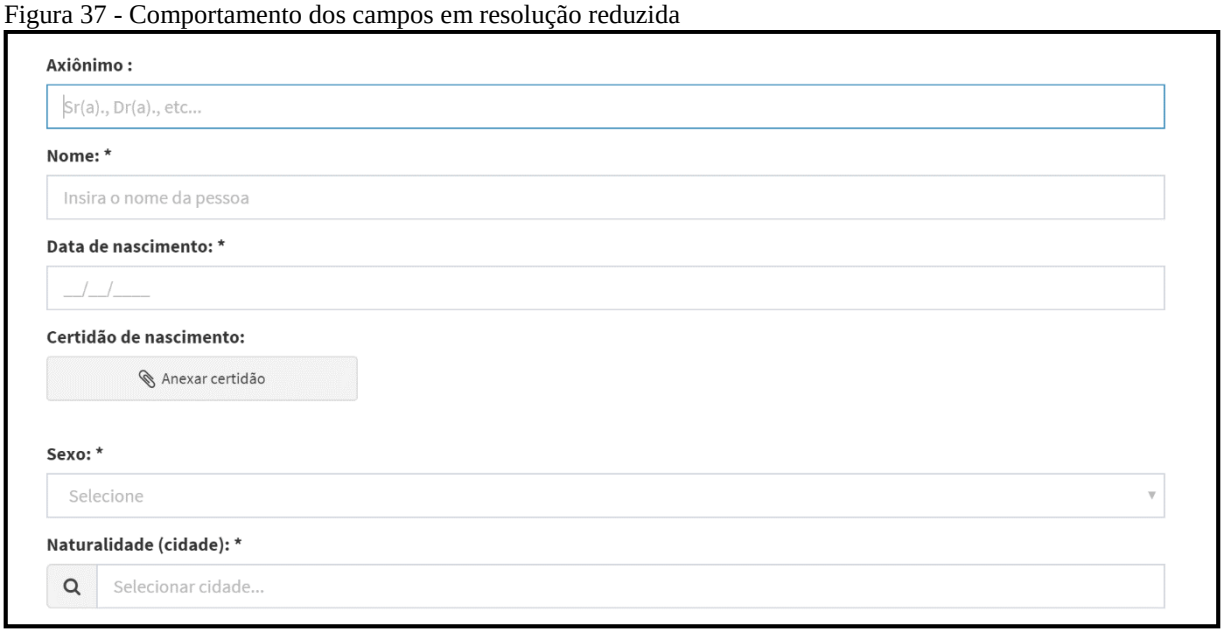
Figura 36 - Validação realizada pelo *plug-in* jQuery *validation*



Fonte: Elabora pelo autor

Conforme mencionado, houve uma atenção especial ao *design* responsivo da aplicação e forma que toda a interface tenta se adaptar aos variados tamanhos de tela. Os campos são configurados para se distribuírem espacialmente um abaixo do outro, melhorando a visualização, como mostra a Figura 37.

Figura 37 - Comportamento dos campos em resolução reduzida

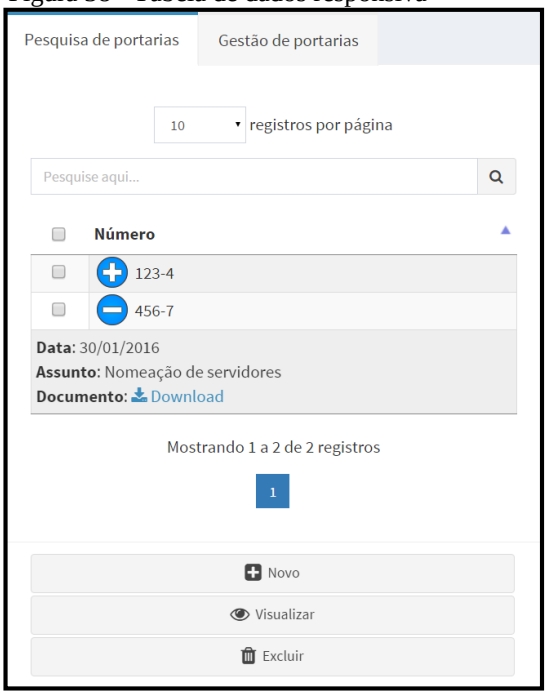


Fonte: Elaborada pelo autor

O elemento mais desafiador de tornar responsivo foi a tabela de dados exibida na aba de pesquisa de registros (FIGURA 33). Embora existam diversos *scripts* de extensão para tornar o *plug-in* DataTables responsivo, muitos apresentam problemas, sendo um dos principais, a

exibição de campos ocultos ao usuário. O usuário Comanche[46] do GitHub criou um *script*[47] baseado no *plug-in* jQuery FooTable[48], totalmente compatível com o DataTables, eliminando os problemas até então encontrados durante o desenvolvimento e tornando a tabela de dados totalmente responsiva. A Figura 38 apresenta o comportamento da tabela de dados em resoluções menores. Na figura observa-se também a configuração diferente dos botões de ações da tabela, aumentando-os de tamanho e exibindo-os verticalmente.

Figura 38 - Tabela de dados responsiva



Fonte: Elaborada pelo autor

As ações de adição, alteração e exclusão nos registros cadastrados são realizadas por *transactions* (transações) de forma que durante alguma destas ações as operações devem ser executadas dentro de um bloco único e indivisível (atômico), sendo desfeitas em caso de falha (*rollback*). Este é o caso de documentos que são anexados no cadastro. As informações dos documentos precisam ser salvas no banco de dados antes do próprio cadastro, pois estes serão referenciados no cadastro através de suas chaves estrangeiras. Seguindo o conceito de transações, o *commit* do cadastro é realizado somente se as informações dos documentos estiverem salvas no banco de dados e não ter ocorrido nenhum erro durante o salvamento/renomeação dos arquivos no servidor. Entretanto, esse mecanismo é mais flexível

[46] Disponível em: <https://github.com/Comanche>. Acesso em jan. 2016.
[47] Disponível em: <https://github.com/Comanche/datatables-responsive>. Acesso em jan. 2016.
[48] *Plug-in* jQuery que permite a criação de tabelas responsivas, não importando a quantidade de colunas. Disponível em: <http://fooplugins.github.io/FooTable/index.html>. Acesso em jan. 2016.

em alguns casos, como os cadastros de pessoas e servidores. Mais detalhes sobre essa diferença de implementação são descritos na seção 4.3.1.

Os cadastros de cargos, memorandos, ofícios, portarias e regimes precisaram de uma classe *Model* especial para mapear corretamente o banco de dados. Isso se fez necessário devido a forma como algumas informações são armazenadas no banco de dados. Na aba de Pesquisa dos registros cadastrados (FIGURA 33), nos cadastros de memorandos, ofícios e portarias por exemplo, a tabela de dados exibida na aba contém a coluna Data. Caso o usuário desejasse pesquisar na tabela alguma das datas, os registros não seriam retornados pelo DataTable corretamente, uma vez que datas não são armazenadas no banco de dados seguindo o formato convencional "dd/mm/AAAA". Para solucionar este problema, criaram-se *views*[49] no banco de dados para servirem como modelo para o DataTables, espelhando as tabelas dos cadastros citados, porém com as colunas seguindo o padrão tradicional, permitindo que o DataTables encontre os registros pesquisados de forma correta. O tutorial de como adaptar um *Model* para mapear uma *view* no banco de dados pode ser encontrado em http://programmingarehard.com/2013/11/10/eloquent_and_views.html/.

Por fim, o Quadro 36 descreve resumidamente cada um dos cadastros implementados.

Quadro 36 - Descrição dos cadastros implementados (continua)

Cadastro	Descrição
Bancos	Possui somente o campo para inserção do nome. É utilizado no cadastro de Pessoas para adição de contas bancárias.
Cargos	Possui os campos para nome e tipo (docente, docente contratado e técnico). É utilizado no cadastro de Servidores para seleção do cargo ocupado.
Cidades	Possui os campos para nome e seleção do Estado. É utilizado no cadastro de Pessoas para adição de naturalidade e endereço.

[49] É uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada. Mais em: <https://pt.wikipedia.org/wiki/Vis%C3%A3o_%28banco_de_dados%29>. Acesso em jan. 2016.

Quadro 36 - Descrição dos cadastros implementados (continuação)

Cadastro	Descrição	
Documentos	**Memorandos**	Possui os campos para número, data, assunto, natureza, setor emissor e setor destinatário, nome do emissor e nome do destinatário, observação e anexo do documento digital.
	Ofícios	Idem memorandos.
	Portarias	Possui os campos para número, data, assunto, observação, publicação e anexo do documento digital. Há duas opções para publicação, Diário Oficial da União e Boletim de serviço. Na primeira os campos número do jornal, data, seção e página são exibidos. A segunda opção exibe os campos mês/ano e localidade (Reitoria ou *campus*).
Estados	Possui os campos para nome, sigla e seleção do País. É utilizado no cadastro de Cidades para seleção do estado.	
Etnias	Possui o campo para nome. É utilizado no cadastro de Pessoas para seleção da etnia.	
Formas de desligamento	Possui o campo para nome. É utilizado no cadastro de Servidores para seleção da forma de desligamento do servidor.	
Formas de ingresso	Possui o campo para nome. É utilizado no cadastro de Servidores para seleção da forma de ingresso do servidor.	
Nacionalidades	Possui o campo para nome. É utilizado no cadastro de Pessoas para seleção da nacionalidade.	
Órgãos expedidores	Possui os campos para nome e sigla. É utilizado no cadastro de Pessoas para seleção do órgão expedidor do RG.	
Órgãos profissionais	Possui os campos para nome e sigla. É utilizado no cadastro de Pessoas para seleção do registro profissional.	
Países	Possui os campos para nome e sigla. É utilizado no cadastro de Estados para seleção do País.	

Quadro 36 - Descrição dos cadastros implementados (conclusão)

Cadastro	Descrição
Pessoas	É o cadastro principal da aplicação. Consiste em cadastrar todas as informações básicas de uma pessoa, como dados pessoais, de contato e de formação. Em vista a quantidade de campos presente neste cadastro e sua complexidade, o mesmo será abordado com mais detalhes na seção 4.3.1.
Regimes	Possui os campos para nome, carga horária e dedicação exclusiva. É utilizado no cadastro de Servidores para seleção do regime de trabalho.
Servidores	Na verdade, se trata do cadastro dos dados funcionais. O usuário seleciona uma pessoa cadastrada e o status em que se encontra a mesma na instituição. Após isso há campos para cadastrar as informações do ingresso e desligamento do servidor, como formas de ingresso e desligamento, cargo, SIAPE, regime de trabalho, categoria, data da posse, data do desligamento entre outros. Não contemplou-se o desenvolvimento *back-end* deste cadastro neste trabalho (justificativa na seção 1).
Setores	Possui os campos para nome, sigla, localização, servidor responsável e servidor substituto. É utilizado no cadastro de Memorandos e Ofícios para seleção do setor emissor e do setor destinatário.
Status	Possui o campo para nome. É utilizado no cadastro de Servidores para seleção do status em que se encontra.

Fonte: Elaborado pelo autor

4.3.1 Cadastro de pessoas

O Cadastro de Pessoas é o principal cadastro do sistema Pipou. Como a aplicação é desenvolvida especificamente para a gestão de pessoas é preciso antes de tudo cadastrar todas as pessoas que compõem o núcleo trabalhista da instituição. Quando se fala em "cadastrar pessoas" entende-se que todos os dados intrínsecos a uma pessoa, como nome, CPF, RG, data de nascimento, sexo, naturalidade, filiação, contas bancárias, endereço etc., devem ser cadastrados para identificar uma pessoa. Este cadastro é a base para se cadastrar as informações do servidor, isto é, os dados funcionais ligados àquela pessoa. Tudo isso servirá para todo o resto da aplicação, como abertura de processos, gerenciamento de histórico funcional, funções

gratificadas, auxílios, afastamentos, férias entre outros, os quais necessitam das informações do servidor.

Por se tratar de um cadastro repleto de campos para preenchimento, dividiu-se a interface de gestão do cadastro em três abas: Dados Pessoais, Dados de Contato e Dados de Formação. Cada aba contém campos específicos ao seu contexto.

A aba de Dados Pessoais contém a maior quantidade de campos, por isso seccionou-a em cinco *accordions*:

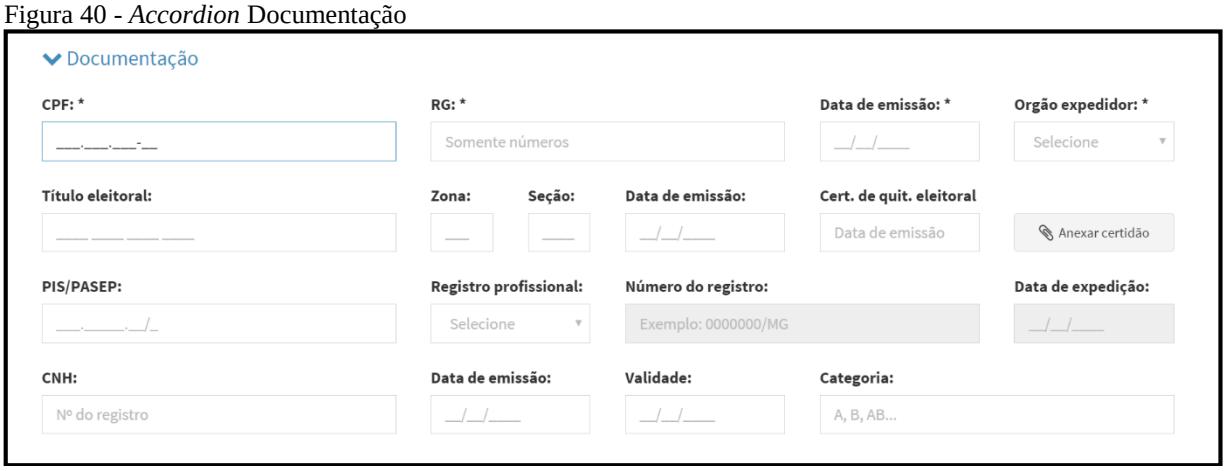
- **Pessoal**: contém os campos referentes aos mais pessoais relacionados à pessoa, tais como axiônimo, nome, data de nascimento, sexo etc. O campo que identifica se a pessoa possui algum tipo de deficiência é habilitado ao marcar o *checkbox* "Possui alguma deficiência?". Acima deste *accordion* é disponibilizado um campo para anexar a foto de perfil (FIGURA 39).

Figura 39 - *Accordion* Pessoal

Fonte: Elaborada pelo autor

- **Documentação**: contém todos os campos relacionados a documentação pessoal, como CPF, RG, título eleitoral, PIS/PASEP, CNH entre outros. Os campos para número do registro e data de expedição são ativados somente após o usuário selecionar algum registro profissional (FIGURA 40).

Figura 40 - *Accordion* Documentação



Fonte: Elaborada pelo autor

- **Estado Civil**: contém campos para seleção do estado civil, nome do cônjuge ou companheiro e anexo para certidão de casamento. No caso quando o estado "Solteiro(a)" é selecionado, os demais campos permanecem desabilitados. Se selecionado um dos demais (Separado(a), Divorciado(a), Viúvo(a)), os campos para nome do cônjuge e anexo para certidão de casamento ficam desabilitados, porém, o campo para nome do companheiro pode ser habilitado marcando um dos *checkboxes* para identificar união estável ou homoafetiva. O campo para nome do cônjuge e o botão para anexo da certidão de casamento são habilitados somente se o usuário selecionar o estado civil "Casado(a)". Neste caso, o *checkbox* para identificar se é uma união homoafetiva também é habilitado, e quando marcado, o campo para nome do companheiro é habilitado, porém a interação com o mesmo é desabilitada, sendo preenchido automaticamente de acordo com a digitação do usuário no campo para nome do cônjuge. Isto é feito de modo que o nome do cônjuge e nome do companheiro sejam persistidos no banco de dados. Embora sejam os mesmos, isso permite identificar na hora de buscar os dados que se trata de um casamento homoafetivo legalizado perante a lei (FIGURA 41).

Figura 41 - *Accordion* Estado civil

Fonte: Elaborada pelo autor

- **Filiação**: contém os campos para nome da mãe e nome do pai (FIGURA 42).

Figura 42 - *Accordion* Filiação

Fonte: Elaborada pelo autor

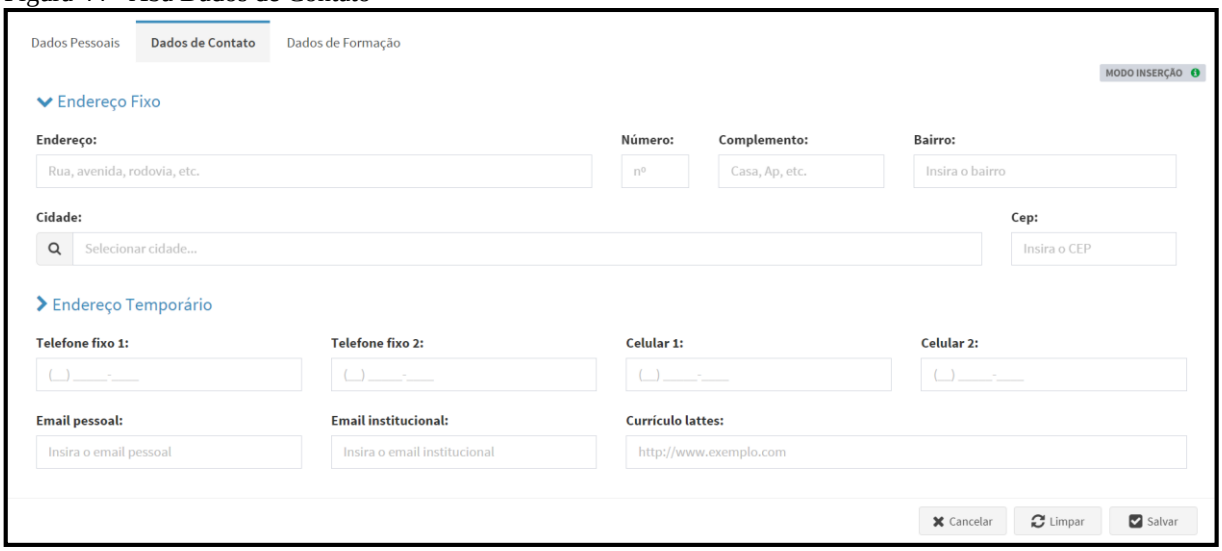
- **Dados Bancários**: contém um botão para adição de conta bancária, a qual possui os campos para buscar um banco cadastrado, agência e número da conta. É possível adicionar várias contas bancárias à pessoa. Pode-se remover uma conta bancária clicando no "x" acima do campo para número da conta (FIGURA 43).

Figura 43 - Accordion Dados Bancários

Fonte: Elaborada pelo autor

Na aba de Dados de Contato encontram-se campos para preenchimento de endereços fixo e temporários (separados por *accordions*), telefones, celulares, e-mails e *link* para currículo lattes, como mostra a Figura 44.

Figura 44 - Aba Dados de Contato



Fonte: Elaborada pelo autor

A aba de Dados de Formação contém seis *accordions*: Técnico, Graduação, Especialização, Mestrado, Doutorado e Pós-Doutorado. Cada um deles possui um botão para adicionar uma formação à pessoa, onde são exibidos os campos para nome do curso, instituição

de ensino e ano de conclusão, e um botão para anexo da documentação comprobatória. Da mesma forma como nas contas bancárias, pode-se remover uma formação clicando no “x”. A Figura 45 apresenta a interface da aba Dados de Formação.

Figura 45 - Aba Dados de Formação

Fonte: Elaborada pelo autor

O cadastro de pessoas é pouco mais flexível no conceito de transações. Neste cadastro existem dados relativos a outras tabelas no banco de dados, como as contas bancárias e formação. Antes de salvar tais registros no banco, é preciso que o registro pessoa já esteja persistido. Neste caso, se ocorrer um erro durante alguma das ações de adição, edição e exclusão sobre estes dados no cadastro de pessoas, não é realizado o *rollback* das operações no registro pessoa, apenas mensagens de erros são exibidas ao usuário informando as ações que não foram executadas corretamente sobre estes dados. Isso faz sentido, uma vez que tais dados não são obrigatórios no cadastro de pessoa e os mesmos recebem a chave estrangeira desta, ou seja, não são salvos diretamente na tabela referente às pessoas no banco de dados.

No caso dos documentos anexados que são ligados à pessoa por meio de chave estrangeira (certidão de nascimento, certidão de quitação eleitoral e certidão de casamento), como não são dados obrigatórios no cadastro, não é feito o *rollback* das operações no registro pessoa se falhar alguma destas nas ações sobre os documentos, também são emitidas mensagens para o usuário. Reforçando, as operações nos registros são desfeitas somente se falhar alguma delas diretamente sobre o mesmo, como por exemplo, no momento de salvar o registro pessoa no banco de dados. Esta diferença em relação aos documentos anexados é a mesma para o cadastro de servidores, onde também é permitido o anexo de documentos não obrigatórios.

Como os cadastros de memorandos, ofícios, portarias, pessoas e servidores permitem o anexo de documentos, fez-se necessário implementar de forma genérica a funcionalidade para

upload e *download* de documentos, de modo que pudesse ser utilizada em todos estes cadastros, sem precisar ficar replicando inúmeras linhas de código que seriam pouco alteradas.

4.4 *Upload* e *Download* de documentos

Para implementar a funcionalidade de *upload* de documentos utilizou-se o *plug-in* Dropzone. Para o caso de *download* utilizou-se o *plug-in* jQuery fileDownload que possibilita transferir um arquivo do servidor via AJAX. Ele permite enviar junto a requisição de *download* parâmetros para serem usados no lado do servidor, como por exemplo o nome do documento armazenado no servidor, nome que será dado ao ser enviado para o cliente, etc.

As funcionalidades de *upload* e *download* estão presentes em todas as telas onde existe exigência ou opção de anexo de documentos, como é o caso dos cadastros de Documentos, Pessoas e Servidores, e do repositório digital de documentos. Uma janela modal é exibida para executar tais operações. No caso dos cadastros, a janela modal exibida é composta por três abas: Documento(s) anexado(s), *Upload* de arquivo(s) e Detalhes do arquivo. A diferença para o repositório de documentos é que o modal não contém a aba "Documento(s) anexado(s)", visto que ao selecionar uma pasta, todos os arquivos contidos nesta são exibidos na forma de tabela em um *box* ao lado do diretório de pastas, de forma semelhante a contida na aba citada. Tais abas podem ser descritas da seguinte maneira:

- **Documento(s) anexado(s)**: esta aba exibe um tabela construída pelo *plug-in* DataTable na qual apresenta informações como nome, tamanho e *link* para *download* dos documentos anexados referentes ao botão de anexo clicado. Além disso, nesta parte é possível selecionar os documentos em anexo e aplicar ações como visualização e exclusão por meio dos botões presentes abaixo da tabela (FIGURA 46).

Figura 46 - Aba Documento(s) anexado(s)

Fonte: Elaborada pelo autor

- ***Upload* de arquivo(s)**: nesta aba está presente o *plug-in* Dropzone, onde um espaço é reservado para clique ou arraste de arquivos para *upload* (FIGURA 47).

Figura 47 - Aba *Upload* de arquivo(s)

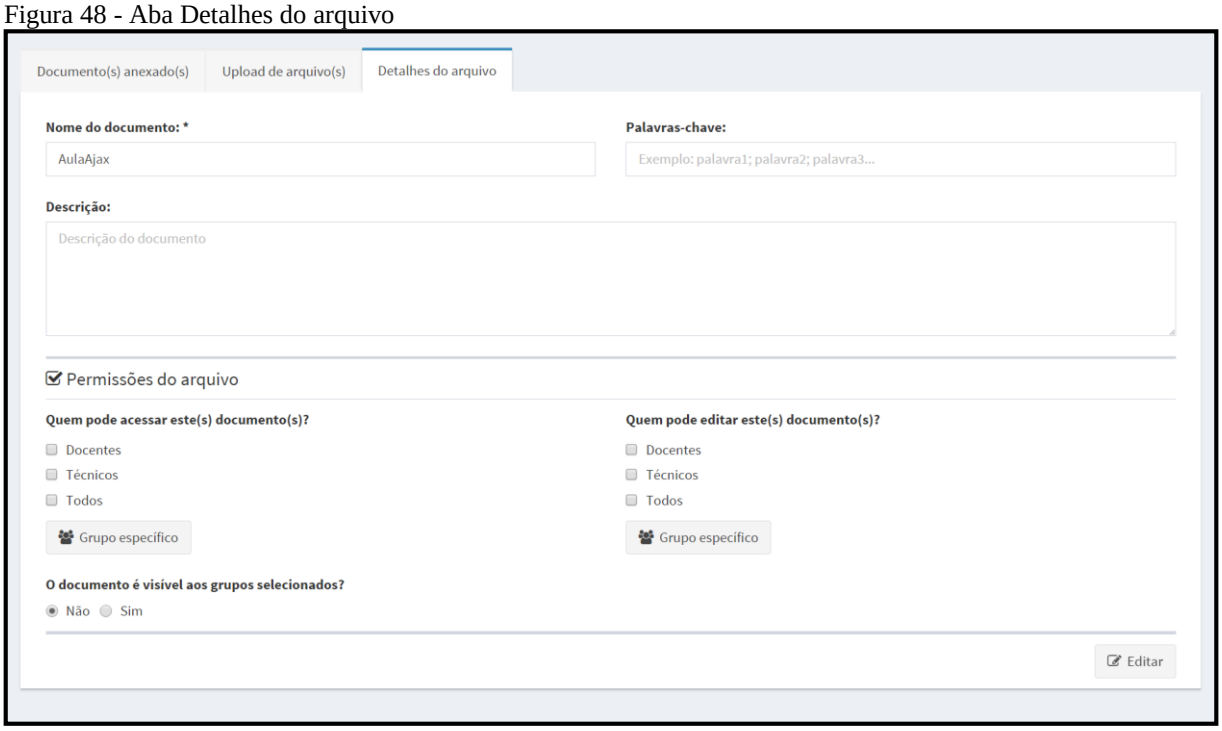
Fonte: Elaborada pelo autor

Conforme observa-se na Figura 47, um ícone gráfico é exibido para identificar o documento de acordo com a extensão do arquivo. Os botões acima deste ícone, da esquerda para a direita, são para executar as ações de iniciar *upload*, visualizar/editar informações do arquivo e excluir arquivo, respectivamente. A mensagem "Upload concluído" embaixo dos dois primeiros documentos destacados na figura indicam que o *upload* dos mesmos foi realizado com sucesso, o que remove o botão de início de upload. Um resumo do documento é exibido ao passar o mouse em cima do ícone, desfocando-o para mostrar as informações. Por ser um elemento gráfico pequeno, o nome do documento é abreviado e mostrado por completo em um *tooltip* abaixo do

ícone. Caso o nome do documento ainda exceda os limites do *tooltip*, o mesmo também será abreviado. Além disso, uma barra exibe a porcentagem de envio do documento para o servidor, a qual é ocultada após a conclusão com ou sem erros do *upload*. Estes últimos detalhes podem ser observados no terceiro documento apresentado na figura.

- **Detalhes do arquivo**: esta aba contém um formulário para visualização e edição de dados do arquivo selecionado, como nome, palavras-chave, observações e permissões de acesso do arquivo. Este último dado, no entanto, ainda não foi implementado funcionalmente, elaborou-se apenas a parte gráfica. Abaixo das permissões do arquivo, está o botão para edição dos dados. Após clicado, um botão para salvar é exibido em seu lugar (FIGURA 48).

Figura 48 - Aba Detalhes do arquivo

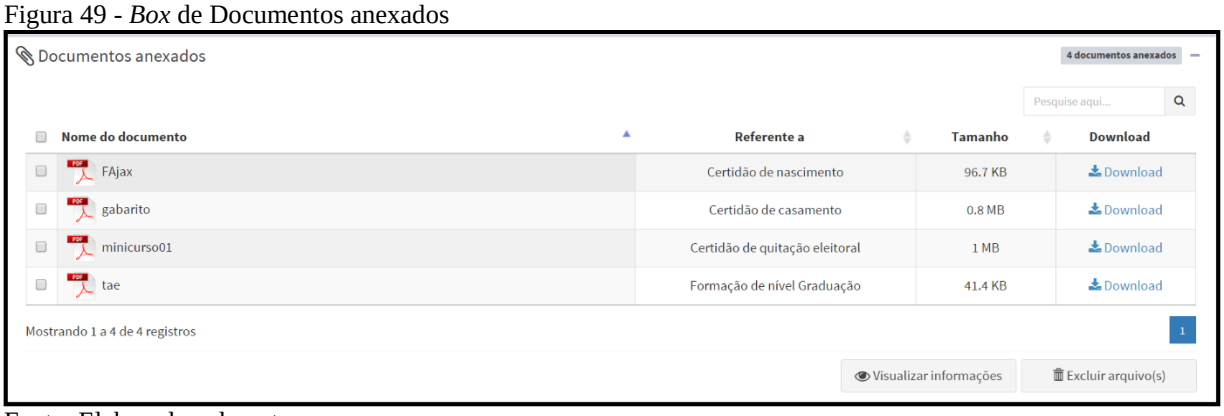


Fonte: Elaborada pelo autor

No caso dos cadastros que contém anexo de documentos, abaixo do *box* que contém a aba de gestão do cadastro está um outro que exibe em forma de tabela todos os documentos anexados referentes àquele cadastro. As funcionalidades são as mesmas encontradas na aba "Documento(s) anexado(s)" do modal de *upload*, porém a tabela possui uma coluna a mais que mostra a referência do documento, isto é, a qual documentação se refere (ex: certidão de nascimento). Embora seja uma funcionalidade replicada, fez-se isso a fim de permitir que o usuário possa visualizar, editar ou excluir qualquer documento sem a necessidade de clicar no

botão ao qual o anexou. A Figura 49 apresenta a interface do *box* que exibe todos os documentos anexados.

Figura 49 - *Box* de Documentos anexados



Fonte: Elaborada pelo autor

Um outro ponto importante está na armazenagem dos arquivos anexados no servidor. Nos cadastros, quando em modo Inserção, todos os arquivos anexados permanecem em uma pasta temporária no servidor até que o cadastro seja salvo, então estes são movidos para a pasta oficial. No modo Edição, qualquer novo documento anexado passa pelo mesmo processo, porém, quando este está salvo na pasta oficial do servidor e é alterado (neste caso, o nome), cria-se uma cópia do mesmo na pasta temporária, e ao salvar o cadastro, move-se a cópia para a pasta oficial, substituindo o arquivo antigo. Pensando na capacidade de armazenamento do servidor, sempre que o usuário atualiza ou muda de página ou sai da aplicação, antes uma requisição via AJAX é enviada ao servidor para que remova todos os documentos anexados referentes ao último cadastro sendo feito que estão na pasta temporária. Isso impede que um acúmulo de documentos seja gerado desnecessariamente na pasta temporária. No caso do repositório de documentos, como não há dados de formulários a serem salvos, todos os arquivos enviados para o servidor vão direto para a pasta oficial do repositório, e qualquer ação de alteração ou remoção é tratada diretamente no arquivo contido nesta pasta, isto é, não há um botão para salvar tudo no final como no caso dos cadastros, toda ação modifica os dados no servidor em tempo de execução.

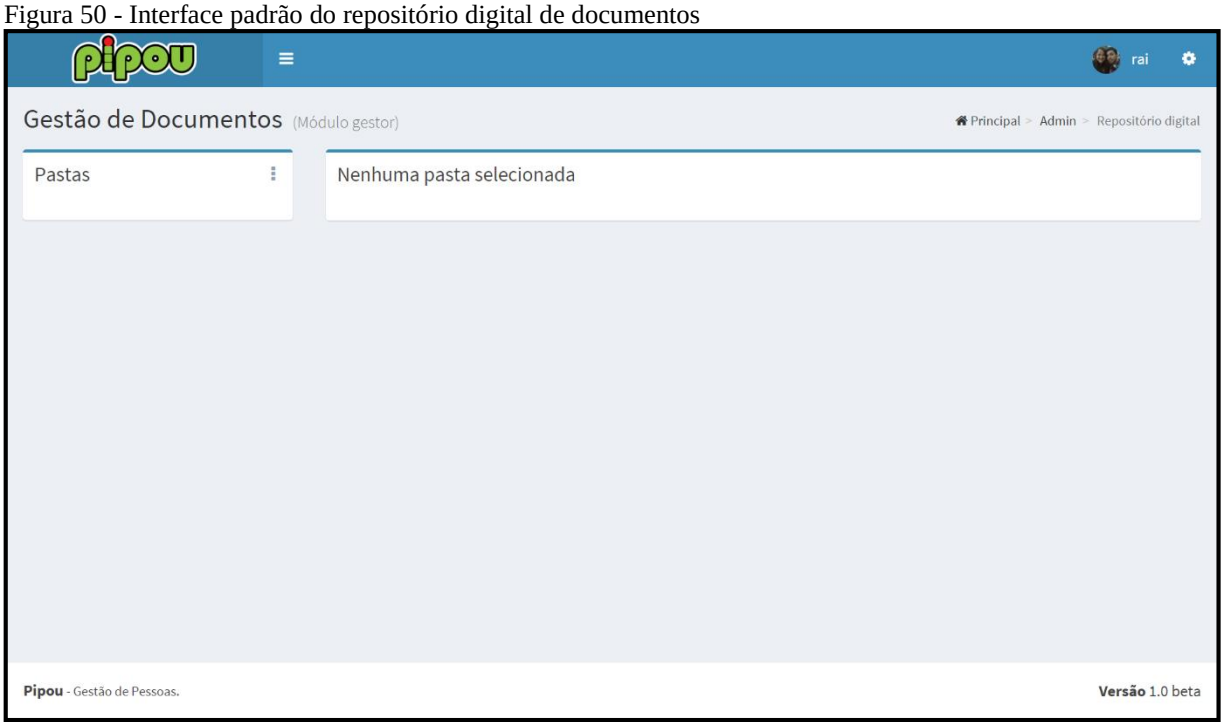
Por fim, a validação dos dados do arquivo como nome, tamanho e extensão é feita no lado do cliente no momento em que o mesmo é enviado ao servidor, via JavaScript, através do próprio *plug-in* Dropzone. No lado do servidor, assim que o arquivo é recebido o mesmo também é validado. A funcionalidade é implementada desta maneira para reduzir o tempo de resposta do servidor. Caso não fosse feito dessa maneira, o envio seria feito somente após o usuário clicar para salvar tudo (no caso particular dos cadastros que possuem formulários

associados ao *upload*), de tal forma que o servidor deveria validar os campos do formulário, receber e validar os arquivos para só então responder com sucesso ou erro. O tratamento de erros seria mais trabalhoso.

4.5 Repositório digital de documentos

Desenvolveu-se o repositório digital da aplicação de maneira simplificada, porém este oferece as funcionalidades básicas de um sistema de arquivos, como criação, renomeação e exclusão de pastas e o gerenciamento de arquivos dentro destas. A Figura 50 mostra a interface padrão do repositório.

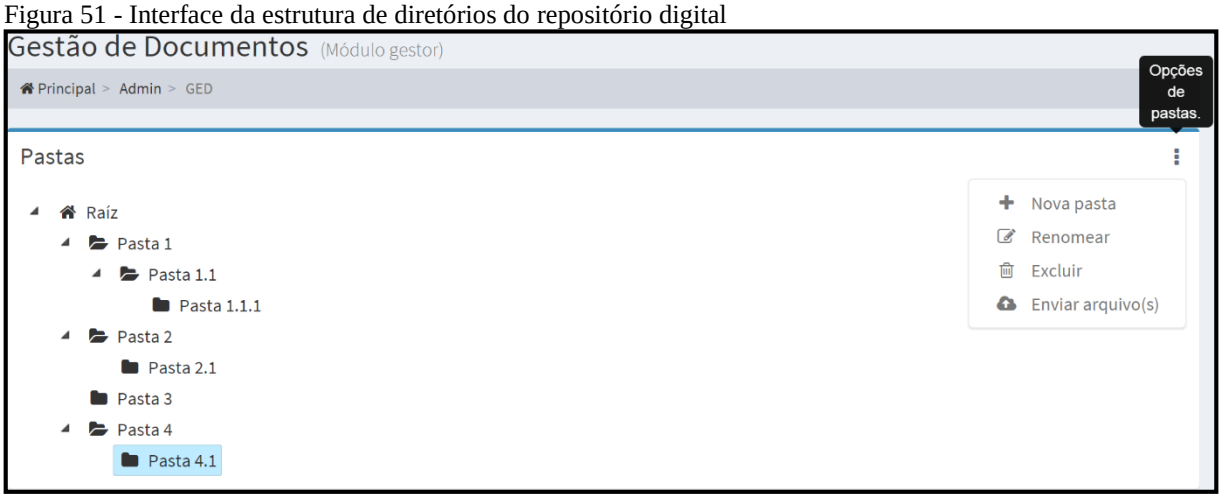
Figura 50 - Interface padrão do repositório digital de documentos



Fonte: Elaborada pelo autor

Com o auxílio do *plug-in* jsTree, um *treeview* é criado para exibir a estrutura de pastas criadas. Na Figura 51 é possível observar a interface desta estrutura, a qual também exibe as opções de pastas que podem ser acessadas pelo botão localizado no canto superior direito ou pelo menu de contexto fornecido pelo *plug-in* aberto quando se clica com o botão direito sobre a pasta.

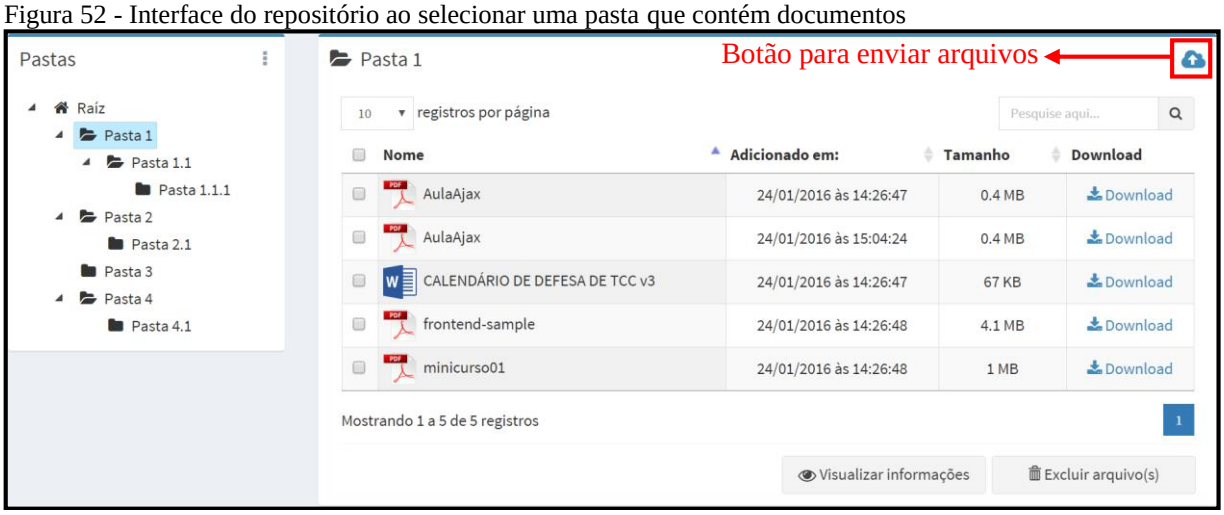
Figura 51 - Interface da estrutura de diretórios do repositório digital



Fonte: Elaborada pelo autor

Ao selecionar uma pasta, uma requisição é feita para o servidor via AJAX, que retorna a quantidade de documentos armazenados na referida pasta. Em seguida, atualiza-se o *box* de exibição dos documentos da pasta selecionada utilizando a quantidade de documentos retornada para manipular a interface. Caso existam documentos na pasta o *box* exibe-os em forma de tabela, sendo esta criada pelo *plug-in* DataTables. Caso contrário, se a pasta está vazia é exibida uma mensagem informando sobre o estado da pasta, disponibilizando um *link* para o envio de documentos. A Figura 52 mostra a interface do repositório quando uma pasta contendo documentos é selecionada. A opção para envio de documentos, presente nas opções de pasta e no menu de contexto, é replicada na forma de um botão para o *box* que exibe a documentação, localizado no canto superior direito, de forma a facilitar e agilizar o acesso a tal funcionalidade.

Figura 52 - Interface do repositório ao selecionar uma pasta que contém documentos



Fonte: Elaborada pelo autor

O repositório permite a exclusão de múltiplas pastas. Para selecionar várias pastas, basta clicar na pasta com a tecla *Ctrl* pressionada. Não é permitido a criação ou renomeação com

mais de uma pasta selecionada, nestes casos uma janela modal com uma mensagem alertando o usuário é exibida. Caso nenhuma pasta esteja selecionada e o usuário tente executar uma das ações (renomeação, exclusão e envio de arquivos) para a mesma, uma janela modal também é exibida alertando-o. Não é permitido se ter mais de uma pasta raiz no repositório, assim, se nenhuma pasta está selecionada e existir uma pasta raiz, se o usuário tentar adicionar uma nova pasta, este será alertado.

Implementou-se o repositório a fim de auxiliar no gerenciamento da documentação da Coordenadoria do SGP do IFMG *campus* Formiga. Seu intuito é imitar o funcionamento básico do sistema de arquivos dos sistemas operacionais, permitindo que o usuário crie pastas e adicione documentos a elas de forma facilitada e intuitiva, tudo dentro da própria aplicação. É claro que espera-se que as pastas e documentos armazenados no repositório sejam de relevância para o objetivo do sistema, a gestão de pessoas, mas esta funcionalidade dá total liberdade ao usuário para usá-la como preferir.

4.6 Métricas do sistema

Muitos arquivos foram gerados durante o desenvolvimento deste trabalho. Mais de 160 são utilizados pela aplicação, entre eles modelos, visões, controladores, estilos, *scripts*, *migrations*, *seeds*, validadores e arquivos PHP a parte, sem levar em conta *plug-ins* JavaScript e jQuery, arquivos dos *frameworks Bootstrap* e Laravel. Os Quadros a seguir apresentam as classes e *scripts* PHP implementados neste trabalho, bem como uma breve descrição de suas funções e a quantidade de linhas de código[50] codificadas. Os Quadro 37, 38, 39, 40 e 41 listam respectivamente os *Models*, *Controllers*, Classes Sementes, *Migrations* e Arquivos a parte.

Quadro 37 - *Models* criados durante o desenvolvimento (continua)

Classe	**Descrição**	**Linhas de código**
Banco	Classe *Model* para mapear a tabela banco no ORM.	12
BaseModel	Classe *Model* para ser herdada pelos outros *models*.	108

[50] Para o cálculo de tal métrica, utilizou-se o *software* CLOC (*Count Lines of Code*). Disponível em: <http://cloc.sourceforge.net/>. Acesso em jan. 2016.

Quadro 37 - *Models* criados durante o desenvolvimento (continuação)

Classe	Descrição	Linhas de código
Cargo	Classe *Model* para mapear a tabela cargo no ORM.	12
Cidade	Classe *Model* para mapear a tabela cidade no ORM.	24
ContaBancaria	Classe *Model* para mapear a tabela conta_bancaria no ORM.	15
DocumentacaoFormacao	Classe *Model* para mapear a tabela documentação_formacao no ORM.	15
Documento	Classe *Model* para mapear a tabela documento no ORM.	66
Estado	Classe *Model* para mapear a tabela estado no ORM.	15
EstadoCivil	Classe *Model* para mapear a tabela estado_civil no ORM.	12
Etnia	Classe *Model* para mapear a tabela etnia no ORM.	12
Formacao	Classe *Model* para mapear a tabela formacao no ORM.	18
FormaDesligamento	*Model* para mapear a tabela forma_desligamento no ORM.	12
FormaIngresso	Classe *Model* para mapear a tabela forma_ingresso no ORM.	12
Log	Classe *Model* para mapear a tabela log no ORM.	12
Memorando	Classe *Model* para mapear a tabela memorando no ORM.	18
Nacionalidade	Classe *Model* para mapear a tabela nacionalidade no ORM.	12
Oficio	Classe *Model* para mapear a tabela oficio no ORM.	18

Quadro 37 - *Models* criados durante o desenvolvimento (continuação)

Classe	Descrição	Linhas de código
OrgaoExpedidor	Classe *Model* para mapear a tabela orgao_expedidor no ORM.	12
OrgaoProfissional	Classe *Model* para mapear a tabela orgao_profissional no ORM.	12
Pais	Classe *Model* para mapear a tabela pais no ORM.	15
Pasta	Classe *Model* para mapear a tabela pasta no ORM.	15
Permissao	Classe *Model* para mapear a tabela permissao no ORM.	15
Pessoa	Classe *Model* para mapear a tabela pessoa no ORM.	63
Portaria	Classe *Model* para mapear a tabela portaria no ORM.	18
Regime	Classe *Model* para mapear a tabela regime no ORM.	12
Rota	Classe *Model* para mapear a tabela rota no ORM.	12
Setor	Classe *Model* para mapear a tabela setor no ORM.	30
Status	Classe *Model* para mapear a tabela status no ORM.	12
TipoFormacao	Classe *Model* para mapear a tabela tipo_formacao no ORM.	12
Usuario	Classe *Model* para mapear a tabela usuario no ORM.	37
ViewCargos	Classe *Model* para mapear a *view* da tabela cargos no ORM.	10
ViewMemorandos	Classe *Model* para mapear a *view* da tabela memorando no ORM.	10

Quadro 37 - *Models* criados durante o desenvolvimento (conclusão)

Classe	Descrição	Linhas de código
ViewOficios	Classe *Model* para mapear a *view* da tabela oficio no ORM.	10
ViewPortarias	Classe *Model* para mapear a *view* da tabela portaria no ORM.	10
ViewRegimes	Classe *Model* para mapear a *view* da tabela regime no ORM.	10
Total		**708**

Fonte: Elaborado pelo autor

Quadro 38 - *Controllers* criados durante o desenvolvimento (continua)

Classe	Descrição	Linhas de código
BancoController	Classe *Controller* para tratar as requisições no cadastro de Bancos.	166
CargoController	Classe *Controller* para tratar as requisições no cadastro de Cargos.	173
CidadeController	Classe *Controller* para tratar as requisições no cadastro de Cidades.	177
DocumentoController	Classe *Controller* para validar os documentos anexados nos cadastros.	145
EstadoController	Classe *Controller* para tratar as requisições no cadastro de Estados.	196
EtniaController	Classe *Controller* para tratar as requisições no cadastro de Etnias.	166
FormaDesligamentoController	Classe *Controller* para tratar as requisições no cadastro de Formas de Desligamento.	166
FormaIngressoController	Classe *Controller* para tratar as requisições no cadastro de Formas de Ingresso.	166
MainController	Classe *Controller* para tratar as requisições na tela principal.	19

Quadro 38 - *Controllers* criados durante o desenvolvimento (continuação)

Classe	**Descrição**	**Linhas de código**
MemorandoController	Classe *Controller* para tratar as requisições no cadastro de Memorandos.	339
NacionalidadeController	Classe *Controller* para tratar as requisições no cadastro de Nacionalidades.	166
OficioController	Classe *Controller* para tratar as requisições no cadastro de Ofícios.	339
OrgaoExpedidorController	Classe *Controller* para tratar as requisições no cadastro de Órgãos Expedidores.	174
OrgaoProfissionalController	Classe *Controller* para tratar as requisições no cadastro de Órgãos Profissional.	174
PaisController	Classe *Controller* para tratar as requisições no cadastro de Países.	174
PessoaController	Classe *Controller* para tratar as requisições no cadastro de Pessoas.	1461
PortariaController	Classe *Controller* para tratar as requisições no cadastro de Portarias.	351
PublicController	Classe *Controller* para tratar as requisições na página inicial (*login*) e para autenticação do usuário.	28
RegimeController	Classe *Controller* para tratar as requisições no cadastro de Regimes.	178
RepositorioController	Classe *Controller* para tratar as requisições no repositório digital de documentos.	456
SetorController	Classe *Controller* para tratar as requisições no cadastro de Setores.	202

Quadro 38 - *Controllers* criados durante o desenvolvimento (conclusão)

Classe	**Descrição**	**Linhas de código**
StatusServidorController	Classe *Controller* para tratar as requisições no cadastro de Status.	166
Total		**5582**

Fonte: Elaborado pelo autor

Quadro 39 - Classes sementes criadas durante o desenvolvimento (continua)

Classe	**Descrição**	**Linhas de código**
BancoSeeder	Classe semente para semear a tabela banco.	139
CargoSeeder	Classe semente para semear a tabela cargo.	43
CidadeSeeder	Classe semente para semear a tabela cidade.	5583
DatabaseSeeder	Classe semeadora principal. Chama a execução das classes sementes.	30
DocumentoSeeder	Classe semente para semear a tabela documento.	45
EstadoSeeder	Classe semente para semear a tabela estados.	44
EtniaSeeder	Classe semente para semear a tabela etnia.	21
FormaDesligamentoSeeder	Classe semente para semear a tabela forma_desligamento.	19
FormaIngressoSeeder	Classe semente para semear a tabela forma_ingresso.	21
NacionalidadeSeeder	Classe semente para semear a tabela nacionalidade.	212
OrgaoExpedidorSeeder	Classe semente para semear a tabela orgao_expedidor.	60
OrgaoProfissionalSeeder	Classe semente para semear a tabela orgao_profissional.	71
PaisSeeder	Classe semente para semear a tabela pais.	209

Quadro 39 - Classes sementes criadas durante o desenvolvimento (conclusão)

Classe	**Descrição**	**Linhas de código**
PermissaoSeeder	Classe semente para semear a tabela permissao.	65
PessoaSeeder	Classe semente para semear a tabela pessoa.	49
RegimeSeeder	Classe semente para semear a tabela regime.	22
RotaSeeder	Classe semente para semear a tabela rota.	38
SetorSeeder	Classe semente para semear a tabela setor.	52
StatusServidorSeeder	Classe semente para semear a tabela status.	19
TipoFormacaoSeeder	Classe semente para semear a tabela tipo_formacao.	20
UsuarioSeeder	Classe semente para semear a tabela usuario.	26
Total		**6788**

Fonte: Elaborado pelo autor

Quadro 40 - *Migrations* criadas durante o desenvolvimento (continua)

Classe	**Descrição**	**Linhas de Código**
create_database	*Migration* para criar a base de dados da aplicação.	850
create_session_table	*Migration* para criar a tabela session no banco para armazenar informações da sessão	17
create_docs_formacao_table	*Migration* para criar a tabela documentação_formacao no banco.	22
create_view_cargos	*Migration* para criar a *view* da tabela cargo no banco.	22
create_view_regimes	*Migration* para criar a *view* da tabela regime no banco.	25
create_view_memorandos	*Migration* para criar a *view* da tabela memorando no banco.	27
create_view_oficios	*Migration* para criar a *view* da tabela oficio no banco.	27

Quadro 40 - *Migrations* criadas durante o desenvolvimento (conclusão)

Classe	Descrição	Linhas de Código
create_view_portarias	*Migration* para criar a *view* da tabela portaria no banco.	23
Total		**1013**

Fonte: Elaborado pelo autor

Quadro 41 – Arquivos a parte criados durante o desenvolvimento

Arquivo	Descrição	Linhas de código
CustomValidator	Classe com métodos de validações personalizadas.	301
Seguranca	Classe com métodos de segunraça para a aplicação.	58
Routes	Arquivo PHP contendo todas as rotas da aplicação.	124
Util	Classe com métodos utilizáveis em algum momento na aplicação, como armazenamento de *log* e geração de *breadcrumbs* por exemplo.	293
Total		**1076**

Fonte: Elaborado pelo autor

Por fim o Quadro 42 resume os quadros supracitados acrescentando arquivos de visão (páginas HTML *.blade.php*), *scripts* (*scripts* jQuery *.js*) e estilo (folhas de estilos *.css*), destacando a quantidade destes bem como o total de linhas de código implementadas.

Quadro 42 - Quantidades de arquivos e linhas de código implementados

Arquivo	Quantidade	Linhas de código
Classes e *Scripts* PHP	90	15167
Visão	33	4339
Script	24	15912
Estilo	16	1672
Total	**163**	**37090**

Fonte: Elaborada pelo autor

5 CONSIDERAÇÕES FINAIS

Todo o desenvolvimento deste trabalho contribuiu veementemente para a agregação de novos conhecimentos no desenvolvimento *web*. Embora a produtividade na criação de

aplicações *web* ainda seja menor se comparada à plataforma *desktop*, a evolução das tecnologias existentes, bem como o surgimento de novas, tem contribuído fortemente para um desenvolvimento cada vez mais rápido, seguro, funcional, elegante e eficaz.

Apesar do estudo de novas tecnologias ter sido a base para a conclusão deste trabalho, muitos conhecimentos adquiridos nas disciplinas do Bacharelado em Ciência da Computação do IFMG *Campus* Formiga foram de suma importância para o desenvolvimento desta versão do sistema de informação Pipou, de onde destacam-se engenharia de *software*, algoritmos e estruturas de dados, desenvolvimento *web*, banco de dados, interface humano-computador, dentre outras.

Como em todo projeto, vários obstáculos e dificuldades surgem no seu decorrer. O domínio de novas tecnologias e o entendimento da legislação e dos processos executados pela gestão de pessoas no âmbito público são exemplos encontrados no desenvolvimento deste trabalho. Por outro lado, algumas facilidades também surgiram durante o desenvolvimento. O fato de ser um sistema inteiramente *web*, usando das tecnologias mais conhecidas e utilizadas, tornou a pesquisa por soluções mais rápida e fácil, visto a vasta quantidade de documentação disponível na internet. Embora os aprendizados da biblioteca jQuery e do *framework* Laravel tenham tomado um tempo considerável, uma vez dominados aumentaram veemente a produtividade de trabalho.

A experiência adquirida no desenvolvimento de *softwares* e a carga de conhecimentos novos agregados a formação foram as principais conquistas deste trabalho. Muito do aqui aprendido poderá ser utilizado em trabalhos futuros, além de facilitar no aprendizado de novas tecnologias para o desenvolvimento *web*.

Por fim, espera-se que o sistema Pipou em sua completude possa complementar as funcionalidades providas pelo atual sistema em uso pelo SGP no IFMG *campus* Formiga, melhorando o processo de interação entre o SPG e os servidores do instituto, facilitando a execução das atividades e a recuperação da documentação sob demanda, reduzindo assim os tempos de atendimento, tornando os serviços prestados pelo setor mais ágeis e eficientes.

5.1 Trabalhos futuros

Muitas funcionalidades e recursos ainda não foram desenvolvidos em virtude da complexidade do projeto completo, da restrição no tamanho da equipe e na disponibilidade de

tempo dos desenvolvedores, devendo ser contemplados em trabalhos futuros. Dentre as melhorias sugeridas para novos trabalhos, destacam-se:

- **Certificado SSL/TSL (HTTPS)**: como a aplicação irá lidar com dados sensíveis de servidores sob o ponto de vista de confidencialidade, é preciso um meio de criptografar toda a comunicação realizada com o sistema. Por se tratar de uma aplicação que se comunica pela rede usando o protocolo HTTP, a melhor opção é o uso de um certificado de segurança SSL/TSL na comunicação entre o navegador e o servidor da aplicação.
- **Migração para as versões mais atuais do jQuery e dos *plug-ins* utilizados, *Bootstrap* e Laravel**: durante o desenvolvimento evitou-se atualizar tais ferramentas para evitar problemas de compatibilidade e afetar o funcionamento da aplicação. Todas essas ferramentas possuem versões mais atualizadas e melhoradas, o que justifica a migração da aplicação para uso das mesmas, sempre utilizando seus melhores recursos para garantir produtividade.
- **Integração com o motor de busca de documentos**: o motor de buscas em documentos é um tema abordado por outro Trabalho de Conclusão de Curso, do aluno Roger Santos Ferreira, membro experiente da equipe de desenvolvimento do sistema Pipou. O objetivo deste motor é realizar a busca em documentos digitais no formato PDF e ordená-los com base em relevância de resultados para o usuário. Será integrado com a aplicação a fim de prover um mecanismo rápido e eficaz de busca por todos os documentos armazenados nesta, seja como exigência documental de cadastros, seja para gestão de documentos do SGP.
- **Implementação dos demais módulos da aplicação**: muitas funcionalidades ainda precisam ser implementadas para atender completamente o SGP do *campus* Formiga, como gerenciamento de processos, históricos funcionais, auxílios, afastamentos, férias, licenças médicas, funções gratificadas, dentre outros enumerados no levantamento de tarefas executadas pelo SGP mencionadas neste trabalho.
- **Hospedagem da aplicação**: inicialmente pretende-se hospedar a aplicação na plataforma OpenShift[51], a qual funciona como um "servidor virtual" que provê

[51] Disponível em: <https://www.openshift.com/>. Acesso em jan. 2016.

serviço de hospedagem gratuita (com espaço para armazenamento limitado) para aplicações desenvolvidas em diversas tecnologias.

- **Testes de funcionamento em dispositivos com tamanho de telas reduzidas**: embora se tenha utilizado recursos para adaptar a interface a variadas resoluções de tela, ainda não testou-se a aplicação em outros dispositivos, como *smartphones* e *tablets*, visto que esta foi inteiramente desenvolvida localmente nos computadores dos desenvolvedores. Desta forma, uma vez que a aplicação esteja hospedada na internet, será possível testar o comportamento da interface em outros dispositivos.
- **Documentação**: após o término do desenvolvimento, toda a documentação será disponibilizada para uso e manutenção do sistema.
- **Implantação**: após a implementação de mais alguns módulos, poderá ser realizada uma implantação inicial da aplicação no SGP do *campus* para uso, coletando *feedbacks* dos usuários sobre *bugs* e possíveis melhorias e adaptações. Com base nos resultados observados na implantação novos recursos poderão ser adicionados à ferramenta.
- **Implementação de um *Optical Character Recognition* (OCR, Reconhecedor Ótico de Caracteres)[52] próprio**: o motor de buscas citado precisa utilizar um OCR proprietário para reconhecer os termos digitados na pesquisa nos documentos. Como os melhores OCRs do mercado são proprietários e suas licenças são caras, pretende-se implementar uma versão própria para uso na aplicação por equipe voluntária ou por novos Trabalhos dede Conclusão de Curso.

[52] Trata-se de uma tecnologia para reconhecer caracteres a partir de um arquivo de imagem ou mapa de bits, sejam escaneados, escritos à mão, datilografados ou impressos. Dessa forma, é possível obter um arquivo de texto editável por um computador.

REFERÊNCIAS

AÇÃO SISTEMAS. A importância do Sistema de RH na gestão de pessoas. **Ação Sistemas de Informática Ltda**, 2012. Disponível em: <http://www.acaosistemas.com/blog/2012/03/21/a-importancia-do-sistema-de-rh-na-gestao-de-pessoas/>. Acesso em: 11 jan. 2016.

ALMEIDA, V. Modelos de Processo de Software Incremental e Iterativo. **ITnerante:** Estudos de TI para Concursos Públicos, 2015. Disponível em: <http://www.itnerante.com.br/profiles/blogs/modelos-de-processo-de-software-incremental-e-iterativo>. Acesso em: 29 dez. 2015.

ASLESON, R.; SCHUTTA, N. **Fundamentos do AJAX**. Rio de Janeiro: Alta Books, 2006. *Apud* LIMEIRA, J. L. S. **Utilização de AJAX no desenvolvimento de sistemas Web.** 2006. 42p. Monografia (Especialização em Web e Sistemas de Informação) - Curso de Especialização em Web e Sistemas de Informação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006. Disponível em: <http://www.limeira.eti.br/monografia_ajax.pdf>. Acesso em: 10 jan. 2016.

AVOYAN, H. As melhores plataformas PHP para 2015. **iMasters**, 2015. Disponivel em: <http://imasters.com.br/linguagens/php/as-melhores-plataformas-php-para-2015/?trace=1519021197&source=single>. Acesso em: 12 jan. 2016.

BAZZOTTI, C.; GARCIA, E. A Importância Do Sistema De Informação Gerencial Na Gestão Empresarial Para Tomada De Decisões-Artigo Publicado na Revista da Universidade Estadual do Oeste do Paraná. **Ciências Sociais Aplicadas em Revista**, v. 6, n. 11, 2006.

BEIGHLEY, L.; MORRISON, M. **Use a Cabeça!:** PHP & MySQL. Tradução de Marcelo Santos. Rio de Janeiro: Alta Books, 2010.

BENEDETTI, R.; CRANLEY, R. **Use a cabeça!:** jQuery. Tradução de Elda Couto. Rio de Janeiro: Alta Books, 2013. 536 p.

BERNARDES, I. P.; DELATORRE, H. **Gestão documental aplicada**. São Paulo: Arquivo Público do Estado de São Paulo, 2008. 54 p. Disponível em: <http://www.arquivoestado.sp.gov.br/site/assets/publicacao/anexo/gestao_documental_aplicada.pdf>. Acesso em: 16 dez. 2015.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3ª. ed. Rio de Janeiro: Elsevier, 2015. 416 p.

BIBEAULT, B.; KATZ, Y. **jQuery in Action**. Manning, 2008.

BOEHM, B. W. Spiral Development: Experience, Principles, and Refinements. **Spiral Development Workshop**, Pittsburgh, 37 p., fev. 2000.

BOEHM, B. W. A Spiral Model of Software Development and Enhancement. **Computer**, v. 21, n. 5, p. 61-72, 1988.

BUCANEK, J. **Learn Objective-C for Java Developers**. Apress, 2009.

CAELUM. **Desenvolvimento Web com HTML, CSS e JavaScript**. Caelum ensino e inovação, 2015. Disponível em: <https://www.caelum.com.br/apostilas/>. Acesso em: 30 jan. 2016.

CCUEC. CSS - Cascading Style Sheets: Estilo e Produtividade para Websites. **EAD CCUEC - Mini Cursos Virtuais**, 2002. Disponível em: <http://www.ggte.unicamp.br/minicurso/css/index.html>. Acesso em: 08 jan. 2016.

CHAFFER, J.; SWEDBERG, K. **Learning jQuery:** Better Interaction Design and Web Development with Simple JavaScript Techniques Packt Publishing, 2007.
CHIAVENATO, I. **Gestão de Pessoas:** o novo papel dos recursos humanos nas organizações. 3ª. ed. Rio de Janeiro: Elsevier, 2010. 624 p.

CHIAVENATO, I. **Gestão de Pessoas:** o novo papel dos recursos humanos nas organizações. 3ª. ed. Rio de Janeiro: Elsevier, 2010. 624 p.

DANGAR, H. **Learning Laravel 4 Application Development:** Develop real-world web applications in Laravel 4 using its refined and expressive syntax. Packt Publishing, 2013.

DA ROSA, D.; DA SILVA, T. L. Adaptação de interfaces para dispositivos móveis com HTML5. In: EATI – 4º Encontro Anual de Tecnologia da Informação, 1., 2013, Frederico Westphalen. **Anais do EATI.** Frederico Westphalen: Novatec, 2013. p. 249-252. Disponível em: <http://www.eati.info/eati/2013/assets/anais/artigo249.pdf>. Acesso em: 05 jan. 2016.

DE SOUZA, L. G. G. **ERP: Principais conceitos, vantagens e desvantagens.** 2005. 46p. Monografia (Graduação em Ciência da Computação) - Curso de Ciência da Computação, Universidade Presidente Antônio Carlos, Barbacena, 2005. Disponível em: <http://www.unipac.br/site/bb/tcc/tcc-a1e2bae285863c7db684d73078938897.pdf>. Acesso em: 20 dez. 2015.

DIAS, O. Banco de Dados no Laravel. **Softerize Magazine**, 2014. Disponível em: <http://magazine.softerize.com.br/tutoriais/php/laravel/banco-de-dados-laravel>. Acesso em: 26 jan. 2016.

EIS, D. Uma breve história do CSS. **TABLELESS**, 2006. Disponível em: <http://tableless.com.br/uma-breve-historia-do-css/>. Acesso em: 07 jan. 2016.

FERREIRA, E.; EIS, D. **HTML5:** Curso W3C Escritório Brasil. São Paulo: W3C Brasil, 2013.

FIGUEIREDO, E. Entendendo o padrão MVC na prática. **TABLELESS**, 2015. Disponível em: <http://tableless.com.br/entendendo-o-padrao-mvc-na-pratica/>. Acesso em: 30 jan. 2016.

FLATSCHART, F. **HTML5:** embarque imediato. Rio de Janeiro: Brasport, 2011.

FRANCO, R. S. T. **Estudo comparativo entre frameworks Java para desenvolvimento de aplicações web:** JSF 2.0, Grails e Sping web MVC. 2011. 90p. Monografia (Especialização em Tecnologia Java) – Programa de Pós-Graduação em Tecnologia, Universidade Tecnológica Federal do Paraná, Curitiba, 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/492/1/CT_JAVA_VI_2010_16.PDF >. Acesso em: 30 jan. 2015.

GAMMA, E. et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2007. 364 p.

GARRET, J. **Ajax:** A New Approach to Web Applications. São Francisco: Adaptive Path, 2005. Disponível em: <https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf>. Acesso em: 09 jan. 2016.

GIL, A. C. **Gestão de pessoas:** enfoque nos papéis profissionais. São Paulo: Atlas, 2007. 312 p.

HIRAMA, K. **Engenharia de software:** qualidade e produtividade com tecnologia. Rio de Janeiro: Elsevier, 2011.

JONES, M. T. Entenda o Representational State Transfer (REST) no Ruby. **Site da IBM**, 2012. Disponível em: <http://www.ibm.com/developerworks/br/library/os-understand-rest-ruby/>. Acesso em: 25 jan. 2016.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de informação gerenciais**. 7ª. ed. São Paulo: Pearson Prentice Hall, 2007.

LEMAY, L. **Aprenda a criar páginas web com HTML e XHTML em 21 dias**. São Paulo: Pearson Education, 2002.

LIMEIRA, J. L. S. **Utilização de AJAX no desenvolvimento de sistemas Web.** 2006. 42p. Monografia (Especialização em Web e Sistemas de Informação) - Curso de Especialização em Web e Sistemas de Informação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006. Disponível em: <http://www.limeira.eti.br/monografia_ajax.pdf>. Acesso em: 10 jan. 2016.

MARCORATTI, J. C. **Desenvolvendo para desktop ou para Web?**, [200-?]. Disponível em: <http://www.macoratti.net/vbn_dkwb.htm>. Acesso em: 05 jan. 2016.

MARCORATTI, J. C. **O processo de Software**, 2005. Disponível em: <http://www.macoratti.net/proc_sw1.htm>. Acesso em: 29 dez. 2015.

MARCORATTI, J. C. **Padrões de Projeto:** O modelo MVC - Model View Controller, 2009. Disponível em: <http://www.macoratti.net/vbn_mvc.htm>. Acesso em: 23 dez. 2015.

MARTINS, M. MVC simples e prático, Parte I. **K19 Treinamentos**, 2011. Disponível em: <http://www.k19.com.br/artigos/mvc-simples-e-pratico-parte-i/>. Acesso em: 30 jan. 2016.

NASCIMENTO, T. Desenvolvendo com Bootstrap 3: um framework front-end que vale a pena! **Blog pessoal ThiagoNasc.com**, 2013. Disponível em: <http://thiagonasc.com/desenvolvimento-web/desenvolvendo-com-bootstrap-3-um-framework-front-end-que-vale-a-pena>. Acesso em: 06 jan. 2016.

OTWELL, T. Laravel 4.2 Documentation. **Site oficial do Laravel**, 2016. Disponível em: <https://laravel.com/docs/4.2>. Acesso em: 26 jan. 2016.

PFLEEGER, S. L. **Engenharia de software:** teoria e prática. Tradução de Dino Franklin. 2ª. ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de Software:** Uma abordagem profissional. Tradução de Ariovaldo Griesi. 7ª. ed. Porto Alegre: AMGH, 2011.

REES, D. **Laravel Code Bright:** Web application development for the Laravel framework version 4 for beginners. Leanpub, 2013.

RODRIGUES, E. G.; MORAIS, R. D. S.; DA SILVA, S. F. **As Práticas de Gestão de Pessoas**. 2010. Disponível em: <http://www.convibra.com.br/upload/paper/adm/adm_2530.pdf>. Acesso em: 29 jan. 2016.

SANDERS, B. **Smashing HTML5:** Técnicas para a nova geração da web. Tradução de Mariana Bandarra. Bookman, 2012. 368 p.

SILVA, M. S. **HTML5:** A Linguagem de marcação que revolucionou a Web. São Paulo: Novatec, 2014a. 320 p.

SILVA, M. S. **Web Design Responsivo:** Aprenda a criar sites que se adaptam automaticamente a qualquer dispositivo, desde desktop até telefones celulares. São Paulo: Novatec, 2014b. 336 p.

SILVA, M. S. **Bootstrap 3.3.5:** Aprenda a usar o framework Bootstrap para criar layouts CSS complexos e responsivos. São Paulo: Novatec, 2015. 225 p.

SOARES, W. **AJAX (Asynchronous JavaScript And XML):** guia prático para Windows. São Paulo: Érica, 2006. *Apud* LIMEIRA, J. L. S. **Utilização de AJAX no desenvolvimento de sistemas Web.** 2006. 42p. Monografia (Especialização em Web e Sistemas de Informação) - Curso de Especialização em Web e Sistemas de Informação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006. Disponível em: <http://www.limeira.eti.br/monografia_ajax.pdf>. Acesso em: 10 jan. 2016.

SOMMERVILLE, I. **Engenharia de Software**. Tradução de Ivan Bosnic e Kalinka G. de O. Gonçalves. 9ª. ed. São Paulo: Pearson, 2011.

SOUZA, F. Uso de frameworks para desenvolvimento web e mitos que já deveriam ter desaparecido. **Profissionais TI**, 2010. Disponivel em: <http://www.profissionaisti.com.br/2010/01/uso-de-frameworks-para-desenvolvimento-web-e-mitos-que-ja-deveriam-ter-desaparecido/>. Acesso em: 30 jan 2016.

SURGUY, M. History of Laravel PHP framework, Eloquence emerging. **MAXOFFSKY - Website of Maksim Surguy**, 2013. Disponível em: <http://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/>. Acesso em: 13 jan. 2016.

TEGON, C. A. **Sistema de informação de recursos humanos no Brasil**. 2006.

TEIXEIRA, F. O que é Responsive Web Design? **Blog de AI**, 2012. Disponível em: <http://arquiteturadeinformacao.com/mobile/o-que-e-responsive-web-design/>. Acesso em: 17 jan. 2016.

TELES, B. A. W.; DE AMORIM, M. R. L. Gestão de Mudança: superando dificuldades na implantação dos Sistemas de Informação nas organizações. **X Simpósio de Excelência em Gestão e Tecnologia-SEGET. Anais eletrônicos**, 2013.

THE PHP GROUP. O que é o PHP? **Documentação oficial do PHP**, 2016. Disponível em: <http://php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 08 jan. 2016.

W3C. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0. **W3C Recommendation**, 2000. Disponivel em: <http://www.w3.org/TR/xhtml1/#xhtml>. Acesso em: 05 jan. 2016.

W3C. Cascading Style Sheets. **Site oficial da W3C**, 2016. Disponível em: <http://www.w3.org/Style/CSS/>. Acesso em: 07 jan. 2016.

W3C BRASIL. **CSS:** Curso W3C Escritório Brasil. São Paulo: W3C Brasil, [2012?]. Disponível em: <http://www.w3c.br/pub/Cursos/CursoCSS3/css-web.pdf>. Acesso em: 07 jan. 2016.

WAZLAWICK, R. S. **Engenharia de Software:** conceitos e práticas. Rio de Janeiro: Elsevier, 2013.

WELLING, L.; THOMSON, L. **PHP e MySQL:** Desenvolvimento Web. Tradução de Edson Furmankiewicz e Adriana Kramer. 3ª. ed. Rio de Janeiro: Elsevier, 2005.

WINSPIRE WEB SOLUTION. 7 Best PHP Frameworks for 2015. **LinkedIn**, 2015. Disponível em: <https://www.linkedin.com/pulse/7-best-php-frameworks-2015-winspire-web-solution>. Acesso em: 12 jan 2016.