INSTITUTO FEDERAL DE SANTA CATARINA

WAGNER CREPES DE SOUZA

CONSTRUTOR DE SISTEMAS WEB

Gaspar

2018

WAGNER CREPES DE SOUZA

CONSTRUTOR DE SISTEMAS WEB

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistema do Câmpus Gaspar do Instituto Federal de Santa Catarina como requisito parcial para aprovação na unidade curricular Trabalho de Conclusão de Curso II.

Orientador: Prof. Me. Andrei de Souza Inacio
Coorientador: Prof. Me. Alexandre Altair de Melo

Gaspar

2018

S729c Souza, Wagner Crepes de
Construtor de sistemas web / Wagner Crepes de Souza ; orientador, Andrei de Souza Inacio, coorientador, Alexandre Altair de Melo, 2018.
53 p.

Trabalho de Conclusão de Curso (graduação) – Instituto Federal de Santa Catarina, Câmpus Gaspar, Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Gaspar, 2018.

Inclui referências.

1. Web. 2. Framework. 3. Banco de dados. 4. Desenvolvimento de softwares. I. Inacio, Andrei de Souza. II. Melo, Alexandre Altair de. III Instituto Federal de Santa Catarina. Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. IV. Título.

CDD 005.3
CDD 005.42

Ficha catalográfica elaborada por Cláudia Kautzmann, CRB 14/1214

AGRADECIMENTOS

Agradeço primeiramente a Deus por me dar forças para enfrentar as dificuldades.

Agradeço a minha esposa Franciele pela paciência e apoio, pelos momentos de alegria e dificuldades durante os estudos.

A meu pai Marcio e minha mãe Marlene pelo incentivo ao estudo e por seus ensinamentos no qual me fizeram chegar até aqui.

A meu irmão Ruan por ter me acompanhado de perto nessa jornada.

A meu grupo de louvor Reviver em Cristo pelo apoio.

A meus professores orientadores Alexandre e Andrei por terem me guiado com sabedoria, auxiliando com correções e instruções para este trabalho.

A meus amigos do curso pelos bons momentos que passamos juntos.

RESUMO

Devido à necessidade de construir sistemas com mais precisão, torna-se cada vez mais importante a utilização de *frameworks* para auxiliar no desenvolvimento de sistemas Web. *Frameworks* definem uma convenção para estruturar as entidades, arquivos de código fonte com base em padrões de projetos. Dessa forma o desenvolvedor deve seguir o padrão de codificação do *framework*, tornando o código mais padronizado. Um projeto estruturado tende a reduzir custos, aumentar a qualidade da aplicação e diminuir o tempo de desenvolvimento. Este trabalho realizou um estudo comparando as principais características de alguns dos principais *frameworks* gratuitos disponíveis para PHP, com o objetivo de construir um novo *framework*, contendo as características mais importantes identificadas nos *frameworks* estudados. Também objetivou construir uma interface, na qual, com apenas alguns passos, o desenvolvedor construirá uma aplicação com persistência em banco de dados sem utilização de linhas de comandos.

Palavras-chave: Web. *Framework*. Banco de dados. Desenvolvimento de *softwares*.

ABSTRACT

Due to the need of building systems more accurately, it becomes increasingly important to use frameworks to assist in the development of WEB systems. Frameworks define a convention for structuring entities, source code files based on project patterns. Therefore the developer should follow the framework coding pattern, making the code more standardized. A structured project tends to reduce costs, increase application quality and decrease development time. This work carried out a study comparing the main features of some of the key free frameworks available for PHP, with the aim of building a new framework containing the most important features identified in the Frameworks studied. It was aimed to build an interface in which, with only a few steps, the developer will build a database-persistent application without using command lines.

Keywords: Web. Framework. Database. Software development.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CGI - Interface Comum de Porta de entrada (*Common Gateway Interface)*
DAO - Objetos de acesso a dados (*Data Access Object*)
CSS - Folha de Estilo em Cascatas (*Cascading Style Sheets*)
HTML - Linguagem de Marcação de Hipertexto (*Hypertext Markup Language*)
HTTP - Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol)*
MVC - Modelo, Visão, Controlador, (Model-View-Controller)
ORM - Mapeamento objeto relacional (*Object-Relational Mappers)*
PDO - Objetos de Dados do PHP (PHP *Data Objects)*
PHP - Pré-processador de hipertexto (PHP: *Hypertext Preprocessor)*
SGBD - Sistema de gerenciamento de banco de dados
SUS - Escala de Usabilidade do Sistema (*System Usability Scale*)
UML - Linguagem de Modelagem Unificada (*Unified Modeling Language*)
XML - Linguagem Extensível de Marcação Genérica (*Extensible Markup Language)*

SUMÁRIO

1 INTRODUÇÃO

Aplicações web estão presentes em várias situações do cotidiano como sistemas de *internet banking*, comércio eletrônico, por exemplo. Estas soluções são executadas em arquiteturas denominadas do tipo cliente-servidor (WINCKLER, PIMENTA, 2002). A parte servidora dessa arquitetura é a responsável por receber as requisições via protocolo HTTP (*Hypertext Transfer Protocol*) processar e retornar dados ao cliente que enviou a requisição.

Tanembaum (2011) relata que a base da Web é a transferência de páginas do servidor para o cliente. Essas páginas enviadas pelo servidor podem conter várias informações desde texto, gráficos, vídeos, ponteiros para outras páginas web entre outros. As páginas mais simples são arquivos armazenados no servidor com conteúdo fixo, chamadas de páginas estáticas. Em empresas e em outras organizações, por vezes essas páginas são construídas para recuperar informações de um banco de dados, de forma a se ter um conteúdo atualizado, obtendo-se, páginas dinâmicas.

Um navegador web é a parte cliente dessa arquitetura. Segundo Tanembaum (2011), o navegador é um programa que exibe uma página web. É a parte da arquitetura que interage com o usuário, captura cliques do mouse em elementos da página exibida, além de ser o caminho para o usuário enviar dados ao servidor por meio de formulários.

O desenvolvimento de uma aplicação web consiste em utilizar várias tecnologias, além de executar um conjunto de tarefas básicas repetitivas (ALVIM, 2010). Por exemplo, em aplicações que consistem em manipular dados de tabelas de um banco de dados relacional, as operações de inclusão, alteração e exclusão serão semelhantes independente da tabela.

Uma dificuldade encontrada em construção de *softwares* conforme descreveu Minetto (2007), é a variedade de formas de programar aplicações com propósitos semelhantes. Dessa forma, uma equipe ao analisar um código produzido por outra, pode levar horas para entender a funcionalidade desse código. Nesse sentido, *frameworks* que visam oferecer uma estrutura para aplicações Web, tem por objetivo melhorar a eficiência, confiabilidade, manutenção e escalabilidade do desenvolvimento de aplicativos (CUI *et al.*, 2009)

Conforme Jobstraibizer (2009), *frameworks* definem uma convenção para estruturar as entidades, arquivos de código fonte com base em padrões de projetos.

Dessa forma o desenvolvedor deve seguir o padrão de codificação do *framework*, tornando o código mais padronizado.

O presente trabalho tem como objetivo a implementação de um *framework* que auxilie na construção de sistemas web, estruturando as classes dos sistemas a serem implementados utilizando padrões de projetos MVC (*Model, View, Controller* ou Modelo, Visão, Controlador) e DAO (*Data Access Object* ou Objetos de acesso a dados). Além de fornecer uma funcionalidade para gerar classes automaticamente, de forma a construir sistemas capazes de manipular desde as informações no banco de dados até a visualização das informações ou dados em tela.

Após essa introdução, será abordado no Capítulo 2 a fundamentação teórica, no Capítulo 3 trabalhos correlatos e no Capítulo 4 descrevendo os materiais e métodos utilizados para a construção do *framework* proposto relatando os resultados obtidos após o desenvolvimento.

1.1 Objetivos

Nesse capítulo serão descritos os objetivos geral e específicos do trabalho.

1.1.1 Objetivo geral

O presente projeto tem como objetivo desenvolver um *framework* para facilitar a construção de sistemas Web. Este *framework* deve poder gerar classes de sistemas conforme o padrão de projeto MVC (*Model*, *View* e *Controller*).

1.1.2 Objetivos específicos

a) Analisar as funcionalidades e as características dos principais *frameworks* existentes para Web;
b) Construir uma interface gráfica para o desenvolvedor cadastrar as entidades, seus atributos e os relacionamentos entre as entidades;
c) Desenvolver um algoritmo para gerar os formulários referentes às entidades cadastradas;
d) Desenvolver um algoritmo que construa as classes de objeto e classes de acesso ao banco dados;

e) Desenvolver os controladores para realizar a comunicação entre interfaces e objetos e armazenamento em banco de dados;
f) Realizar teste finais com profissionais da área afim de medir a usabilidade dos componentes.

1.2 Justificativa

Conforme Teixeira (2008), as empresas investem em *frameworks* para reduzir o tempo de desenvolvimento e melhorar a qualidade de seus produtos. Com a adoção de um *framework* de desenvolvimento, as pessoas envolvidas no projeto tendem a focar seus esforços no entendimento de requisitos do sistema, ao invés de despender um grande esforço na digitação de código.

Com a utilização do *framework* evita-se a programação de tarefas repetidas, conforme Belém *et al*. (2014), uma das principais características do *framework* é o reaproveitamento de códigos oferecendo componentes prontos e tornando o trabalho mais produtivo.

A decisão em utilizar um *framework* segundo Lisboa (2010), reside na necessidade de estruturar os projetos de *software*, devido à grande complexidade que os mesmos alcançaram. Um projeto estruturado tende a reduzir custos, aumentar a qualidade da aplicação, diminuir o tempo de desenvolvimento e facilitar a manutenção.

A motivação em construir um novo *framework* veio da necessidade de obter um *framework* com funcionalidades capazes de construir um sistema Web utilizando interface gráfica para modelar as entidades e seus relacionamentos, diferenciando-se de outros *frameworks* disponíveis, que são abordados no Capítulo 3. Estes uma vez se valem de linhas de comando para executar essas tarefas, de forma que o desenvolvedor ao utilizar esses *frameworks* terá que estudar os comandos que seus componentes oferecem.

Com o *framework* proposto neste trabalho, o desenvolvedor terá várias atividades automatizadas. Primeiramente será necessário cadastrar as entidades que serão persistidas no banco de dados e seus atributos. Para cada atributo dessa entidade será possível definir mascaras e validações para evitar erros de digitação do usuário etc. O *framework* gerará também os arquivos de código fonte no padrão de projetos MVC, para automatização de criação de telas.

Sem a utilização do *framework* o desenvolvedor gastaria esforço na digitação

do código fonte dessas camadas para cada entidade impactando negativamente na produtividade. Com a utilização do *framework* proposto, todo o trabalho será realizado via um *wizard* com assistente gráfico, de forma que o *framework* se encarregará de gerar o código fonte automaticamente após o fornecimento dos dados necessários.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos e as definições de algumas das mais importantes tecnologias disponíveis na atualidade para a construção de sistemas ou aplicações Web.

2.1 Aplicações web

A Web é um serviço que permite o acesso a conteúdo espalhados em milhões de máquinas na internet. Iniciou em 1989 no CERN (*European Center for Nuclear Research*), a proposta para uma teia de documentos veio do físico CERN Tim Berners-Lee. A ideia inicial era compartilhar experimentos de física de partículas em alguns países, por meio de relatórios, plantas, desenhos, fotos ente outros documentos conforme (TANEMBAUM, 2011).

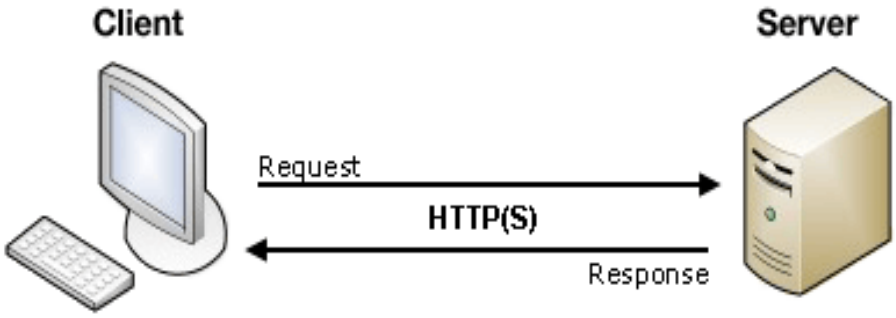
Segundo Pressman (2016), nos primórdios da internet os sites eram formados por páginas estáticas utilizando-se de arquivos com código HTML (*Hypertext Markup Language*) que apresentavam informações usando texto e gráficos limitados. Com a evolução da linguagem HTML e o surgimento de linguagens de desenvolvimento para internet com capacidade de gerar conteúdo HTML dinamicamente, por exemplo as linguagens Java e PHP, nasciam então sistemas e aplicações baseados na Web, as páginas deixaram de ser estáticas e passaram a ser dinâmicas. As aplicações Web evoluíram e foram integradas aos bancos de dados corporativos e às aplicações de negócio. Winckler e Pimenta (2002) relatam que durante este processo de evolução, o número de usuários cresceu exponencialmente e a Web tornou-se acessível a todas as pessoas com uma variedade de aplicações.

2.1.1 A arquitetura cliente-servidor

O serviço Web é executado sobre uma arquitetura denominada cliente/servidor instalado sobre uma rede de computadores heterogênea que são compostas por equipamentos distintos no qual podem operar com sistemas operacionais distintos (WINCKLER; PIMENTA, 2002). Do lado do cliente está o navegador no qual é o intermediário entre o usuário e servidor e disponibiliza uma interface para o usuário informar o endereço do servidor e os formulários para fornecer os dados solicitados pelo servidor.

No lado do servidor são executados os programas CGIs (*Common Gateway Interface*) que recebem a entrada de dados do usuário e realizam a integração com aplicações de banco de dados. A comunicação entre o cliente e o servidor é do tipo *request/response* (pedido/resposta) utilizando o protocolo HTTP. Nesta comunicação o cliente envia um *request* ao servidor solicitando um arquivo. Em seguida, o servidor devolve um *response* ao cliente com o documento e encerra a conexão. A cada pedido que o cliente realizar, uma nova conexão será estabelecida com o servidor não tendo vínculo com a requisição anterior. Por esse motivo o HTTP é considerado um protocolo sem estado, ou seja, *stateless*. A Figura 1 ilustra a comunicação descrita.

Figura 1 - Modelo cliente/servidor



Fonte: MOZILLA, 2017, não paginado.

A Figura 1 ilustra um cliente enviando um *request* ao servidor. O servidor ao receber a solicitação envia um *response* ao cliente e encerra a comunicação.

2.1.2 Linguagens de programação para Internet

Para o desenvolvimento de sistemas Web é necessário além da linguagem de marcação HTML utilizar uma ou mais linguagens de programação, tais como: PHP, *Python*, *JavaScript*, C#, Java entre outras (SILVA, 2011). Como as *tags* HTML são estáticas, cabe a uma linguagem de programação a criação do conteúdo dinâmico que é exibido no navegador web.

PHP é a abreviação de PHP: *Hypertext Preprocessor* ou em tradução livre “Pré-processador de Hipertexto”. É uma linguagem interpretadora de *scripts,* por sua vez são um conjunto de instruções no qual executam as funções dos sistemas. O PHP

possui código fonte aberto e tem como objetivo primário a geração de conteúdo dinâmico para páginas da internet. Isto quer dizer que é possível escrever HTML com o código PHP embutido para gerar algum conteúdo dinâmico. Outra característica do PHP é que o mesmo é executado no lado do servidor, ou seja, seu código-fonte não será relevado ao usuário, que apenas terá acesso ao HTML resultante (MELO; NASCIMENTO, 2007).

JavaScript é uma linguagem de *script* utilizada juntamente com o HTML, criada pela Netscape e Sun Microsystems. Foi uma das primeiras linguagens de scripts da web. É geralmente utilizada em conjunto com HTML para ajudar na interação com usuários e para realizar algumas tarefas do lado cliente, como, por exemplo, mostrar mensagens para usuário quando um campo não foi devidamente preenchido (MELO; NASCIMENTO, 2007).

CSS (*Cascading Style Sheets*) (Folha de Estilo em Cascatas), é uma linguagem utilizada para adicionar estilos em páginas web. Por exemplo: fontes, cores, espaçamentos, etc. (SILVA, 2015).

2.2 Framework

Conforme Cui *et al.* (2009), o *framework* oferece uma estrutura para aplicações web, separando os dados, visualizações e controle em camadas, permitindo obter acoplamentos soltos, melhorando assim a eficiência, confiabilidade, manutenção e escalabilidade do desenvolvimento de aplicativos.

Segundo Alvim (2010, p. 12),

> O *framework* é um conjunto de classes que colaboram entre si proporcionando melhores práticas de desenvolvimento e diminuição à repetição de tarefas. Além disso, evita variações de ‘soluções diferentes para um mesmo tipo de problema’. O que facilita a reutilização e customização dos códigos.

As características de um *framework* que proporcionam benefícios aos desenvolvedores conforme descreveram Fayad e Schmidt (1997) são:

- Modularidade: Encapsulam os detalhes da implementação visível melhorando a modularidade.

- Reusabilidade: Definem componentes genéricos que podem ser reutilizados para criar novas aplicações. Um *framework* é uma estrutura reusável, uma aplicação semipronta e pode ser especializado para construir aplicações específicas.
- Extensão: Seus métodos são adaptáveis pois permitem a extensão das interfaces. Fornece métodos que podem ser implementados para cada aplicação específica.
- Inversão de fluxo controle: Métodos construídos pelos usuários são invocados dentro do *framework*, quem decide em chamar o método é o *framework* e não a aplicação.

2.2.1 Classificação dos Frameworks

Fayad e Schmidt (1997) classificam o escopo dos *frameworks* em três grupos: *frameworks* da infraestrutura do sistema, *frameworks* de integração de módulos e *frameworks* de aplicações empresariais.

Os *Frameworks* da infraestrutura do sistema simplificam o desenvolvimento da infraestrutura do sistema. É utilizado na fase do projeto do *software*. Tratam de questões como comunicação, sistemas operacionais, interfaces gráficas e linguagens de programação. São utilizados internamente pela equipe do projeto e não são vendidos diretamente a clientes (RÉ, 2002).

Frameworks de integração de módulos são usados para integrar aplicações e componentes distribuídos. Projetados para modularizar, reutilizar e estender a infraestrutura de *software* para funcionar em ambientes distribuídos.

Frameworks de aplicações empresariais dirigem-se a várias aplicações que definem um mesmo domínio. São fundamentais para atividades de negócios empresariais.

2.3 Banco de dados

Navathe (2010), afirma que os bancos de dados desempenham um papel crítico em quase todas as áreas que utilizam computadores, incluindo negócios, comércio eletrônico, engenharia, medicina, ente outras. O banco de dados é uma coleção

de dados relacionados que representam algum aspecto do mundo real. Os dados são organizados logicamente coerente com algum significado inerente. Dados são fatos conhecidos que podem ser armazenados, exemplo nomes, telefone e endereço de pessoas, lugares ou coisas.

2.3.1 SGBD – Sistema de gerenciamento de banco de dados

Um sistema gerenciador de banco de dados ou SGBD, é uma coleção de programas que permite o usuário gerenciar o processo de definição, construção, manutenção e compartilhamento de bancos de dados (NAVATHE, 2010). O SGBD disponibiliza uma interface para seus clientes inserir, alterar, excluir e consultar dados armazenados na base de dados.

Conforme Navathe (2010), os primeiros SGBDs comerciais foram disponíveis no início da década de 1980, com o sistema SQL/DS utilizado no sistema operacional MVS da IBM e o SGBD da Oracle. Desde então o modelo relacional foi implantado em uma variedade de sistemas comerciais como por exemplo, SQLServer da Microsoft, DB2 da IBM, etc. Além dos SGBDs comerciais, surgiram os SGBDs de código fonte aberto como MySQL e PostgreSQL.

As seções a seguir apresentam outras funções importantes fornecidas pelo SGBD, que incluem, Controle de Transações, Integridade, Controle de concorrência e recuperação e tolerância a falhas.

2.3.1.1 Controle de transação

É o conjunto de operações a serem executadas. Ao iniciar uma transação, várias operações podem ser executadas no banco de dados. Porém, se alguma falhar todas as outras serão descartadas (*Roll Back*). Caso o conjunto for executado com sucesso todas operações serão salvas permanentemente (*Commit*) (SANTOS *et al.*, 2010).

2.3.1.2 Controle de concorrência

Um SGBD permite múltiplos usuários acessarem o banco de dados. O controle de concorrência torna possível usuários distintos alterar dados em comum ou dados distintos simultaneamente. Antes de existir o SGBD esse controle era realizado pelo sistema operacional, porém cada usuário detinha o direito de gravação para si, e somente quando terminasse, outros acessos de gravação poderiam ser realizados (MILANI, 2006).

2.3.1.3 Recuperação e tolerância a falhas

Uma falha leva ao Banco de Dados a um estado não consistente, as quais podem ocorrer devido à falta de energia, problemas no disco, entrada de dados incorretamente entre outros. Em falhas mais graves consideradas catastróficas, a recuperação ocorre restaurando uma cópia (*backup*) mais recente da base. Em falhas não catastróficas o SGBD desfaz as transações que causaram inconsistência (ASSIS, [201-?]).

2.3.1.4 Integridade

São definidos regras e procedimentos para garantir as características dos dados. Pode-se, por exemplo, definir uma regra para bloquear a inserção de um novo registro com valores repetidos ou também nulos. Este controle é denominado integridade de dados (MILANI, 2006).

Uma característica do banco de dados, conforme Milani (2006), é existir uma chave única primária para cada tabela do banco de dados, recurso este conhecido como integridade de entidade.

Em banco de dados relacionais as tabelas se relacionam por meio da chave estrangeira. Ao preencher um campo de chave estrangeira o SGBD valida se o valor preenchido existe na tabela relacionada. Este recurso é conhecido como integridade referencial (MILANI, 2006).

2.3.2 Modelo relacional

Conforme Navathe (2010), o modelo de dados relacional foi introduzido em 1970 por Ted Codd da IBM Research. O modelo relacional representa o banco de dados como uma coleção de relações, cada relação é considerada uma tabela de valores, a cada linha na tabela representa valores de dados relacionados. Uma tabela representa uma entidade do mundo real. O nome das tabelas e colunas são usados para interpretar o significado de cada linha. Por exemplo, no Quadro 1 contém uma tabela do banco de dados denominada PESSOA, pois a cada linha e coluna são representados fatos sobra uma entidade de pessoa.

Quadro 1 - Exemplo de uma tabela de Pessoa

PESSOA		
Nome	**Data_nascimento**	**Numero_telefone**
João	01/02/1991	00000-0000
Pedro	10/08/1980	99999-9999

Fonte: elaborado pelo autor (2018).

A tabela do banco de dados pode conter uma chave primária (*primary key*) simples ou composta, essa chave é utilizada como identificador único da tabela. A chave primária simples é representada por apenas uma coluna, já uma chave primária composta é representada por mais de uma coluna. No exemplo do Quadro 2, a tabela ESTADO possui o campo Codigo_estado sendo uma coluna chave primária. A coluna Nome pode conter duas linhas com valores iguais, porém não pode haver registro repetidos ou nulos para a coluna chave primária.

Quadro 2 - Exemplo de Banco de Dados armazenando a tabela de Estado

ESTADO	
Codigo_estado (chave primária)	**Nome**
1	SC
2	RS

Fonte: elaborado pelo autor (2018)

Uma tabela se relaciona com outra por meio da chave estrangeira (*foreign key*). Essa chave estrangeira é a referência para chave primária de outra tabela. Conforme

mostrado no Quadro 3, a tabela CIDADE possui a coluna Codigo_estado representando o relacionamento com a tabela ESTADO do Quadro 2.

Quadro 3 - Exemplo de Banco de Dados armazenando a tabela de Cidade

CIDADE		
Codigo_cidade *(chave primária)*	**Nome**	**Codigo_estado (Chave estrangeira)**
1	Ilhota	1
2	Itajaí	1

Fonte: elaborado pelo autor (2018).

2.4 Linguagem de Modelagem Unificada (UML)

Conforme Larman (2007), a UML é uma linguagem visual para especificar, construir e documentar artefatos do sistema por meio de uma notação diagramática padrão para desenhar ou apresentar figuras relacionadas a *softwares,* principalmente orientados a objeto.

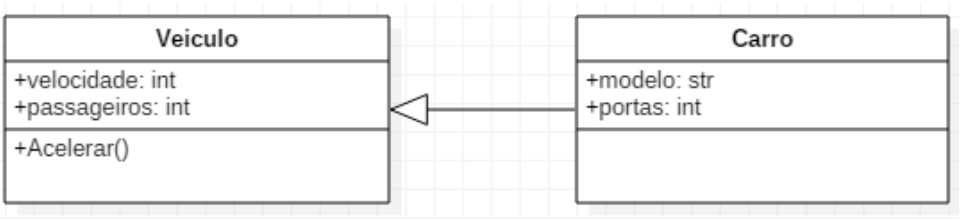
2.4.1 Diagramas de classes

Diagramas de classes são utilizados na UML para ilustrar as classes, interfaces de suas associações. São usadas para modelagem estática do objeto (LARMAN, 2007).

O diagrama representa a classe por meio de um retângulo contendo o nome do objeto, seus atributos e seus métodos. As associações entre objetos são representadas com as seguintes características:

a) Herança: Ocorre quando uma classe filha assume as características de uma classe pai. É representada por uma linha sólida com uma seta fechada conforme ilustrado na Figura 2.

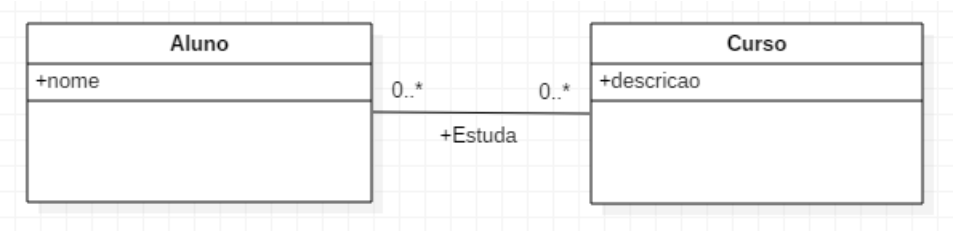
Figura 2 - Interação herança entre duas classes



Fonte: elaborado pelo autor (2018).

b) Associação bidirecional: Representada por uma linha reta entre as classes. Nesse relacionamento as duas classes se conhecem. A Figura 3 ilustra essa associação.

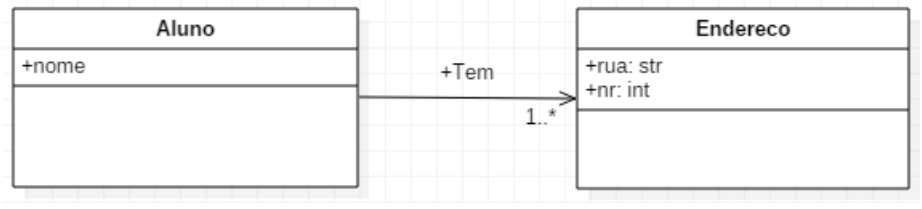
Figura 3 - Associação bidirecional



Fonte: elaborado pelo autor (2018).

c) Associação unidirecional: Representada por uma linha sólida com uma seta aberta conforme a Figura 4. Nesse relacionamento, apenas umas das classes conhece o relacionamento.

Figura 4 - Associação unidirecional



Fonte: elaborado pelo autor (2018).

2.4.2 Diagramas de atividades

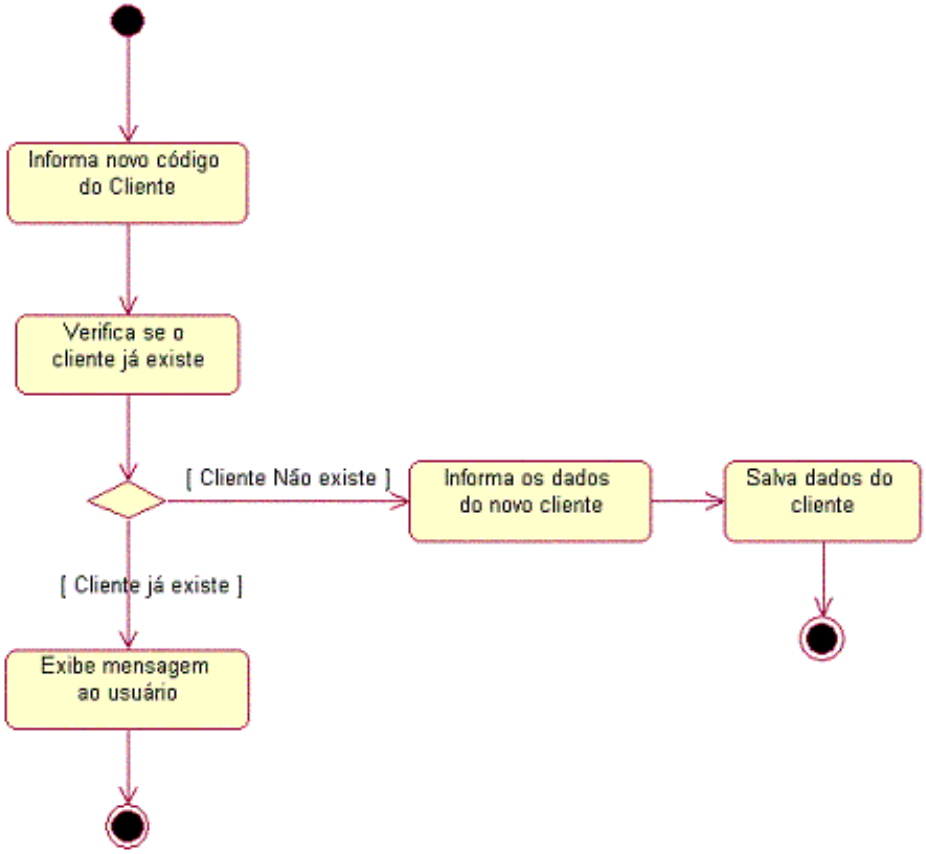
Um Diagrama de atividades UML, segundo Larman (2007, p. 483), “[...] mostra atividades sequenciais e paralelas em um processo. Eles são úteis para modelagem de processos de negócios, fluxo de trabalhos (*workflows*), fluxo de dados e algoritmos complexos”.

Conforme Ventura (2016), os elementos utilizados no diagrama são:

a) *Activity*: Ilustra a atividade propriamente dita.
b) *Partition*: Ilustra fronteira entre módulos, funcionalidades, sistemas ou subsistemas, conforme o nível de detalhe do diagrama (VENTURA, 2016).
c) *Decision*: Representa uma decisão que pode desviar o fluxo do processo.
d) *Initial*: Esse é o primeiro elemento do diagrama definindo o início do fluxo do processo.
e) *Final*: É o elemento que encerra o fluxo do processo.
f) *Fork*: Divide a direção do fluxo.
g) *Join*: Realiza a união de várias direções em uma.

A Figura 5 ilustra um diagrama de atividades.

Figura 5 - Diagrama de atividades



Fonte: MITTELSTEDT, 2012, não paginado.

2.5 Padrões de projetos

Conforme Gamma *et al.* (2008, p. 20), “um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado objetos reutilizável”.

Padrões de projetos são soluções obtidas por meio de experiências reais que foram implementadas nas principais linguagens de programações orientadas a objetos tais como C++ e Smalltalk (GOMMA *et al.*, 2008).

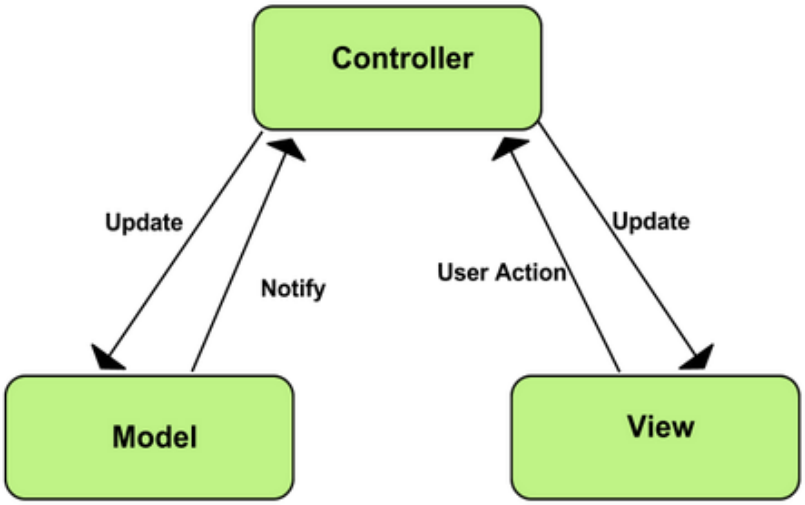
Gamma *et al.* (2008) descrevem 23 padrões de projetos que podem ser aplicados no desenvolvimento de *software*, cada padrão de projetos focaliza um problema ou tópico particular do projeto que podem variar de acordo com a situação ou restrição do projeto.

2.5.1 Padrão de projetos MVC

Segundo Lamim (2009), MVC (*Model View Controller*) é um padrão de arquitetura de *software* que realiza a separação entre as tarefas de acesso aos dados, lógica de negócio e *layout* do sistema, por meio das camadas *model-view-controller*.

A camada *Model* contém a estrutura da entidade, seus atributos e métodos de acesso. Na camada *View* estão os códigos para gerar a interface gráfica, contendo o formulário de entrada dos dados, tabela para listar os registros cadastrados e filtros para restringir a busca na tabela. Já a camada *Controller* contém as ações do sistema, realiza a interação entre o usuário e os métodos das entidades. Recebe as entradas do usuário e envia para as classes de acesso ao banco persistir as informações (LAMIM, 2009). A Figura 6 ilustra as interações entre essas camadas.

Figura 6 - Padrão MVC



Fonte: DANNER, [201-?], não paginado.

2.5.2 Padrão de projetos DAO

O padrão de projetos DAO (*Data Access Object*) (Objetos de acesso a dados), implementa o mecanismo de acesso necessário para trabalhar com fonte de dados. A fonte de dados pode ser um serviço externo como, por exemplo, banco de dados, arquivos, etc. Essencialmente, o DAO atua como um adaptador entre o componente e a fonte de dados (ORACLE, [201-?]).

O padrão DAO oculta os detalhes de implementação de acesso a base de dados de forma que as demais classes não precisam saber desses detalhes o que permite mais segurança.

2.6 Usabilidade de software

Segundo Nielsen (1993) a Usabilidade é um requisito de qualidade de *software* que avalia a facilidade de aprendizado do uso de uma interface, o quanto ela é intuitiva e o esforço necessário para os usuários aprender a utilizar o sistema por meio de alguns fatores que são:

- Facilidade de aprendizagem;
- Facilidade de memorização;
- Eficiência;
- Segurança no uso;
- Satisfação do usuário.

Conforme Cybis (2003), existem diferentes técnicas para avaliar a usabilidade de um sistema:

1. Técnica prospectivas: Busca opiniões de usuários utilizando questionários e entrevistas.
2. Técnica preditiva: Realiza avaliações visando prever erros de interfaces.
3. Técnicas objetivas: Utiliza observação e ensaio de interação entre o usuário e o sistema.

2.6.1 System Usability Scale (SUS)

Para validação da usabilidade do *framework* desenvolvido, adotou-se o questionário SUS (*System Usability Scale),* que segundo Teixeira *et al*. (2017) é um dos métodos mais conhecidos e simples de averiguação do nível de usabilidade de um sistema.

SUS é questionário utilizado para medir percepções de usabilidade de qualquer tipo de interface criado por John Brooke em 1986. O questionário e composto por 10

perguntas com 5 opções de resposta (TEIXEIRA *et al*., 2017). Figura 7 demonstra o formato de resposta e o questionário completo é apresentado no Anexo A.

Figura 7- Formato de resposta do questionário SUS

Strongly Disagree 1	2	3	4	Strongly Agree 5
○	○	○	○	○

Fonte: SAURO, 2009, não paginado.

Para calcular a pontuação final é subtraído 1 das respostas selecionadas das perguntas ímpares e para as respostas selecionadas das perguntas pares é calculado 5 menos o item selecionado. Por fim, é realizada a soma de todas as pontuações e multiplicado por 2,5. Se o resultado for menor que 68 pontos, a interface apresenta sérios problemas de usabilidade.

3 TRABALHOS CORRELATOS

Este capítulo apresenta *frameworks* disponíveis na Internet, para *download* gratuito, que auxiliam no desenvolvimento de sistemas web. Os *frameworks* a serem abordados são *Framework* Yii, *Framework* Zend, *Framework* CodeIgniter. A escolha destes *frameworks* se deu por meio do trabalho de Belém *et al.* (2014), no qual os mesmos foram selecionados utilizando-se dos critérios de popularidade, tamanho da comunidade, suporte, documentação e disponibilidade de recursos.

Além dos selecionados por Belém *et al.* (2014), será abordado sobre o Doctrine, o que é capaz de construir um banco de dados a partir de objetos do PHP, sendo que todos possuem licença gratuita.

3.1 Framework Yii

Yii é um *framework* de alta performance para linguagem PHP que utiliza componentes para o desenvolvimento de grandes aplicações Web. Permite a máxima reutilização de códigos na programação Web e pode acelerar significativamente o processo de desenvolvimento. O nome Yii (pronunciado i) representa as palavras fácil (*easy*), eficiente (*efficient*) e extensível (*extensible*) (BELÉM *et al.*, 2014).

O *framework* Yii é um *software* livre, lançado sob os termos da seguinte licença BSD.

Para utilizar o *framework* Yii é necessário possuir no mínimo um servidor Web com suporte a PHP 5.1.0 e conhecer práticas de programação orientado a objetos. O *framework* Yii possui uma ferramenta de geração de código chamada Gii, essa ferramenta é capaz de gerar código automaticamente criando as classes de *Views, Helper, Controller* e Model.

Abaixo a lista de recursos oferecidos:

- Arquitetura Modelo-Visão-Controlador;
- Objetos de Acesso à Base de Dados (DAO);
- Formulário, validação e suporte a Ajax embutido;
- Autenticação embutida;
- A ferramenta de geração de código do Yii, Gii, acelera o desenvolvimento focando no MVC;

- Console;
- Opções de tema;
- Suporte a cache;
- Segurança;
- Integração com outros *frameworks*;
- Extensões;
- Manipulação de erros.

3.2 Framework *Zend*

O Zend *Framework* é uma coleção de pacotes desenvolvidos em PHP. Teve seu início em março de 2006. É orientado a objetos e usa o padrão de projetos MVC, possui contribuidores de *software* livre. O framework Zend é um *software* livre que utiliza a licença BSD New.

Conforme Chase (2006), os componentes disponibilizados pelo Zend *framework* incluem.

- Zend_*Controller*: Módulo que fornece o controle geral para o aplicativo. Converte pedidos solicitados pelo cliente em ações específicas e assegura que sejam executados;
- Zend_Db: Esse componente é baseado em PHP Data Objects (PDO) e fornece acesso ao banco de dados;
- Zend_Feed: Facilita o consumo de alimentações RSS e Atom;
- Zend_Filter: Fornece funções de filtragem de string, exemplo isEmail() para validar se o dado fornecido pelo usuário é um email válido;
- Zend_InputFilter;
- Zend_HttpClient: Executa pedidos HTTP;
- Zend_Log: Fornece funções para criar logs do sistema por exemplo logs de falhas ou alertas;
- Zend_Mail: Componente para envio de e-mails;
- Zend_Mime: É utilizado pelo Zend_Mail para decodificar mensagens MIME;
- Zend_Pdf: Esse possibilita criar ou editar documentos PDF;
- Zend_Shearch: Esse possibilita executar procuras de palavras em texto;

- Zend_Service_Amazon, Zend_Service_Flickr e Zend_Service_Yahoo: Fornece acesso a APIs de serviço da Web;
- Zend_View: Manipula as telas da página Web;
- Zend_XmlRpc: Facilita a criação de um cliente XML-RPC.

3.3 Framework CodeIgniter

É um *framework* PHP para auxiliar desenvolvedores que precisam de um conjunto de ferramentas simples para criar aplicações da Web completos. Permite que se mantenha o foco no projeto minimizando a quantidade de código necessário para uma determinada tarefa. O *framework* CodeIgniter é um *software* gratuito que utiliza a licença MIT.

Conforme Andrade (2011) os recursos oferecidos pelo CodeIgniter incluem:

- Sistema baseado na *abordagem Model-View-Controller*;
- Extremamente leve;
- Classes completas de banco de dados com suporte a várias plataformas;
- Suporte a *Active Record Database;*
- Validação de Formulários e Dados;
- Segurança e Filtragem XSS;
- Gerenciamento de Sessão;
- Classe de envio de e-mail. Suporte a Anexox, formatação HTML / Texto Puro, múltiplos protocolos (*sendmail*, SMTP, e Mail);
- Biblioteca de Manipulação de Imagens (recorte, redimensionamento, rotação etc.). Suporte a GD, ImageMagick e NetPBM;
- Classe para Upload de Arquivos;
- Classes FTP;
- Localização;
- Paginação;
- Encriptação de Dados;
- *Benchmarking;*
- Cache completo da página;
- Log de Erros;

- Perfis de Aplicação;
- Scaffolding.;
- Classe Calendário;
- Classe User Agent;
- Classe Zip Encoding;
- Classe de Gerador de Template;
- Classe Trackback;
- Biblioteca XML-RPC;
- Classe de Teste de Unidade;
- URLs amigáveis aos motores de busca;
- Roteamento URI Flexível;
- Suporte a Ganchos, Extensões de Classe e Plugins;
- Grande biblioteca de funções "helpers".

3.4 Framework Doctrine

Conforme Andrade (2011), Doctrine é um mapeador de objeto-relacional (*Object-Relational Mappers*) ou (ORM) para PHP. O projeto iniciou em 2006, porém a primeira versão foi lançada em setembro de 2008.

Doctrine fornece persistência para objetos PHP em bancos de dados relacional através de uma camada de abstração (DBAL). As tabelas do banco de dados são representadas pelas classes do PHP e cada registro da tabela é representado por uma instância dessa classe, as colunas da tabela são representadas por atributos da classe correspondente.

Uma de suas principais características é a opção de escrever consultas de SQL utilizando o dialeto DQL *Doctrine Query Language*. Ao executar um comando DQL as instruções são convertidas para instruções do SQL e, ao retornar um registro do banco, o mesmo é transformado em uma instância da classe correspondente a tabela.

Segundo Andrade (2011), Doctrine está dividido em três pacotes principais:

- Common – Possui componentes reutilizáveis;
- DBAL – Este pacote contém uma camada de abstração de banco de dados que facilita a manipulação dos dados;
- ORM – Persiste objetos em uma estrutura de banco de dados relacional;

3.5 Comparações entre os Frameworks

Este capítulo demostra um quadro contendo as principais características dos *frameworks* correlatos abordados e do novo *framework* proposto. O Quadro 4, demonstra as características.

Quadro 4 – Comparativo das características de alguns Frameworks

	Gera código fonte em MVC	**Constrói tabelas do banco de dados**	**Paradigma orientado a objetos**	***Wizard* de configuração com interface gráfica**
Framework Yii	Sim	Não	Sim	Não
Framework Zend	Sim	Não	Sim	Não
Framework CodeIgniter	Sim	Não	Sim	Não
Framework Doctrine	Não	Sim	Sim	Não
Framework Desenvolvido	Sim	Sim	Sim	Sim

Fonte: elaborado pelo autor (2018).

4 DESENVOLVIMENTO E RESULTADOS

Neste capítulo, são apresentados os materiais e métodos adotados para este trabalho e os resultados obtidos após sua validação.

4.1 O Framework

O *framework* é constituído de uma coleção de diretórios e classes organizados em camadas conforme os padrões de projetos MVC e DAO. Nas classes disponíveis estão as funcionalidades para automatizar a construção de novos sistemas. Essas funcionalidades são acessadas por meio de uma página web. Essa página é utilizada para modelar as entidades e seus atributos e relacionamentos.

Esta solução foi baseada na linguagem de programação PHP e no banco de dados MySQL para a sua construção. A interface gráfica exibida no navegador foi desenvolvida utilizando-se HTML e CSS que servem para organizar, formatar e estilizar o visual das páginas web. Essa interface possui alguns ícones como, por exemplos: lápis para simbolizar a funcionalidade ‘editar’ e a lixeira para simbolizar o a exclusão. Esses ícones são vetores disponibilizados gratuitamente pela Font-Awesome que são úteis para adicionar novas fontes à página web. Nas validações de entrada de dados, máscaras dos formulários entre outras interações com o usuário foram utilizadas a linguagem de programação JavaScript com o auxílio da biblioteca jQuery que visa simplificar o uso do JavaScript com funções prontas.

Com base nessas tecnologias e no referencial teórico, o *framework* desenvolvido tem por objetivo disponibilizar componentes para gerar automaticamente códigos PHP, *JavaScript* e SQL além de construir tabelas no banco de dados.

Após descrever as tecnologias utilizadas nesse projeto, aborda-se, nas seções 4.1.1 e 4.1.2 as técnicas de modelagem de *software* utilizadas e as seções seguintes descrevem os requisitos necessários para utilizar o *framework* e sua estrutura e como utilizar seus componentes por meio de uma interface gráfica.

4.1.1 Diagrama de classes

Essa seção apresenta o diagrama de classes utilizado para modelar a estrutura do *framework* implementado. Figura 8 ilustra as classes utilizadas.

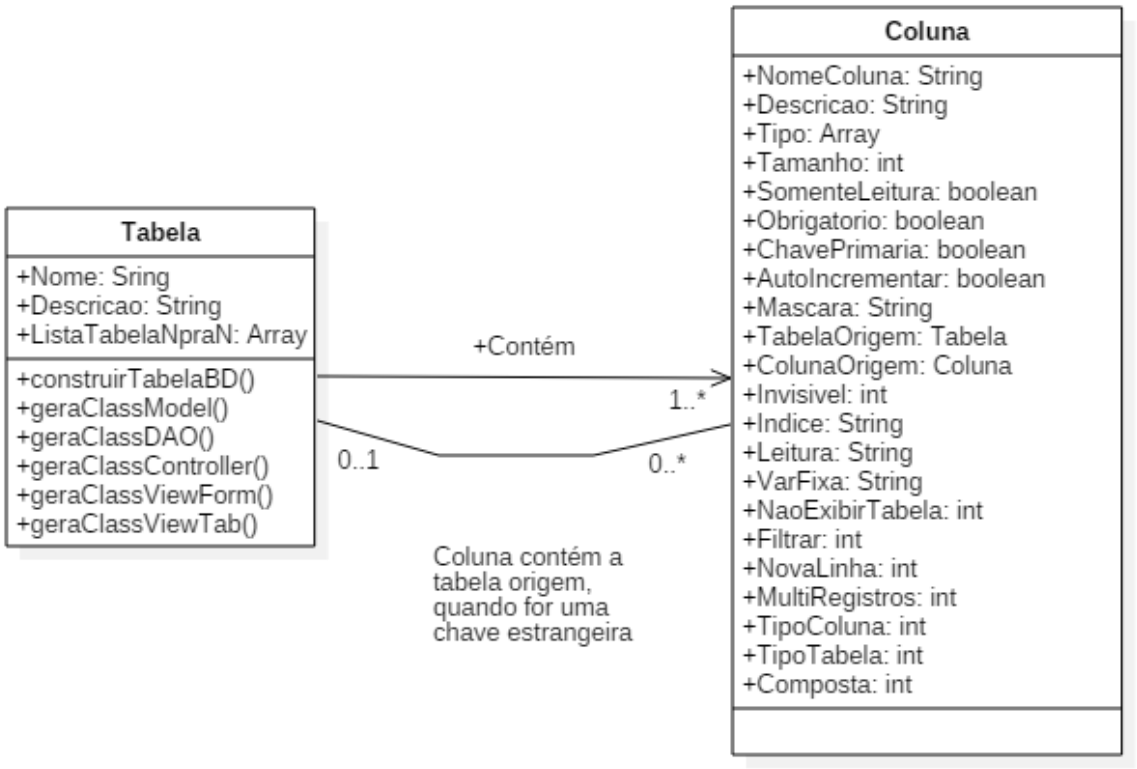
A classe *Tabela* é constituída pelos atributos nome, descrição, ListaTabelaNpraN e por métodos. O atributo nome representa o nome da entidade e será utilizado na construção de tabelas do banco de dados e da estrutura de arquivos do sistema. Nesse caso, não é permitido inserir caracteres especiais e espaços em branco pois as linguagens de programação e os SGBDS não permitem esses caracteres. A descrição é o nome formal da entidade podendo ser utilizado, neste caso, caracteres especiais e espaços. O atributo ListaTabelaNpraN servirá para guardar as entidades que contêm o relacionamento entra a entidade de origem e entidade de destino. Sendo assim, o *framework* gerará uma interface onde para cada registro da tabela de origem, será possível adicionar registros da tabela de destino em uma lista.

A classe *Coluna* é constituída de atributos não tendo métodos, pois a classe Tabela será responsável pelas regras de negócio do *framework* implementado.

O relacionamento entre a classe *Tabela* e a classe *Coluna* é unidirecional, pois a classe tabela contém uma lista de *Colunas*, nesse caso um objeto da classe Coluna só existirá quando um objeto da classe *Tabela* for instanciado.

Quando uma coluna for do tipo Tabela será necessário informar a tabela de origem. Nesse caso a Coluna conhece a Tabela de origem e essa coluna se torna uma ou a chave estrangeira da tabela de destino. Caso houver mais de uma coluna compondo uma chave primária na tabela de origem, o *framework* criará as outras colunas que estão associadas à chave estrangeira na tabela de destino automaticamente referenciando a coluna que as gerou como ColunaGerouComposta.

Figura 8 - Diagrama de classe

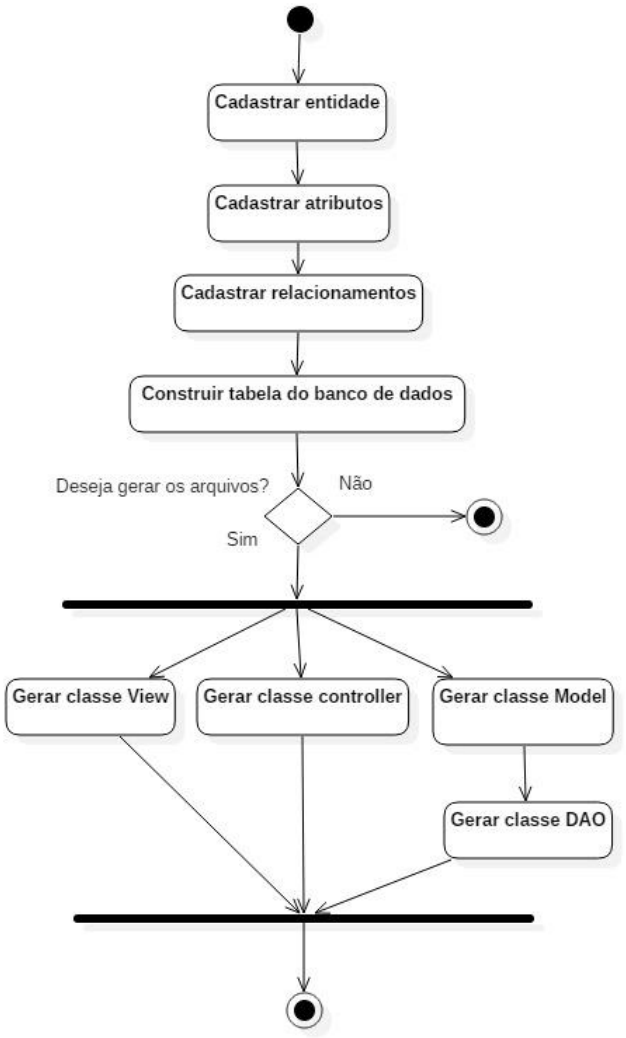


Fonte: elaborado pelo autor (2018).

4.1.2 Diagrama de atividades

Essa seção descreve o diagrama de atividades para modelar o fluxo de dados do *framework* proposto. O *framework* proposto poderá ser utilizado tanto para construir tabelas no banco de dados como também o sistema para o usuário interagir com essas tabelas por meio de alguns passos, conforme ilustra a Figura 9.

Figura 9 - Diagrama de atividades do fluxo de dados



Fonte: elaborado pelo autor (2018).

4.1.3 Requisitos mínimos

O *framework* requer o servidor apache com PHP versão 5.2.0 ou superior. Além do servidor utilizado para executar o *Framework* é necessária a instalação do banco de dados MySQL.

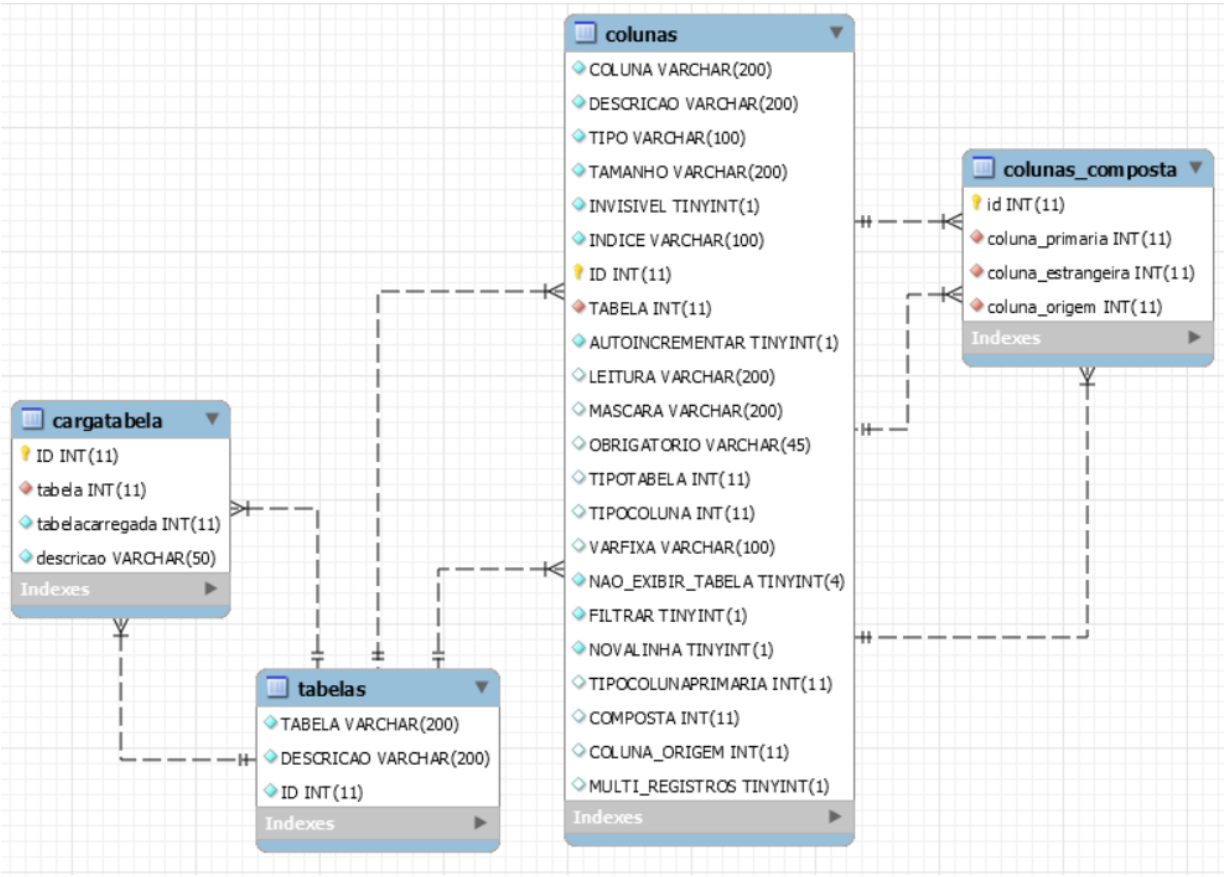
As configurações de conexão de acesso ao banco de dados são informadas no arquivo “db_config.txt” localizado no diretório “config” da estrutura do *framework* conforme apresentada na seção 4.1.5. O banco de dados utilizado pelo *framework* é criado automaticamente ao realizar o primeiro acesso a interface gráfica, conforme mencionada na seção 4.1.4.

4.1.4 Banco de dados do framework

O banco de dados do *framework* é utilizado para armazenar as entidades que serão utilizadas pelos componentes no momento de gerar os arquivos.

Quando o *framework* estiver em uso, será verificado se existem as tabelas representadas na Figura 10 no banco de dados. Caso as mesmas não existirem, serão criadas automaticamente.

Figura 10 - Banco de dados do Framework



Fonte: elaborado pelo autor (2018).

4.1.5 Estrutura do framework

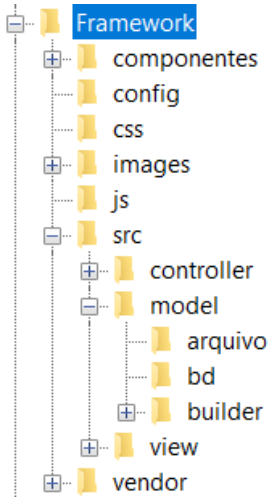
A estruturação foi realizada com o intuito de criar diretórios para organizar os arquivos de CSS, *JavaScript* e as classes dos componentes do *framework*. A Figura 11 demonstra o resultado dos diretórios construídos.

Os diretórios são:

1. componentes – Contém componentes abordados na seção 4.1.6.

2. config – Contém os arquivos de configurações utilizado pelos componentes.
3. css – Contém os arquivos que adicionam estilo em páginas web.
4. images – Local para armazenar as imagens utilizadas pelos componentes do *framework*.
5. js – Contém os arquivos de *JavaScript*
6. src – Constituído por subdiretórios seguindo o padrão de projetos MVC e DAO que servem para organizar as responsabilidades das classes das novas aplicações construídas por estes componentes.
7. vendor – Todas as bibliotecas de terceiros são armazenadas nesse diretório.

Figura 11 - Diretórios utilizados pelo Framework



Fonte: elaborado pelo autor (2018).

4.1.6 Componentes do framework

O *framework* possui os seguintes componentes:

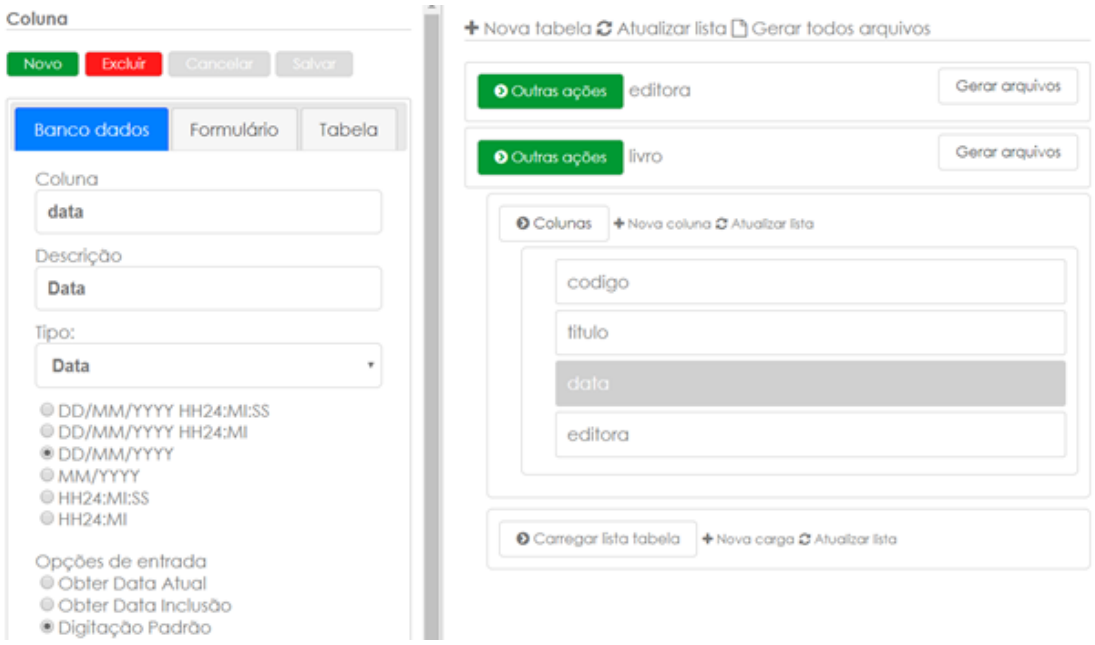
1. Gerador de classes PHP - Esse componente é responsável por gerar as classes PHP *model* e *controller* de uma entidade.
2. Construtor de interfaces - Componente utilizado para criar os arquivos HTML com formulário, filtros de pesquisa e tabela para listar registro de uma entidade.

3. Construtor de tabelas - Esse componente é utilizado no momento em que uma entidade está sendo armazenada no banco de dados do *framework*. Nesse momento é criada a tabela e as colunas da entidade sendo criada.
4. Gerador de menus – Busca todas as entidades cadastradas e gera um menu vertical com os controladores dessas entidades.

4.1.7 Interface gráfica

A interface é acessada no navegador informando o IP ou nome do domínio do servidor Apache acrescida do nome do diretório onde está o *framework*, exemplo: “localhost/framework/componentes/construtor/”. Essa interface inicial foi desenvolvida para possibilitar de forma fácil o cadastro de entidades, colunas e relacionamento entre entidades conforme demostra Figura 12.

Figura 12 - Interface gráfica do framework



Fonte: elaborado pelo autor (2018).

À esquerda da interface são carregados os formulários e na direita está a lista de tabelas. Quando for clicado em uma tabela a baixo dessa tabela será expandido uma lista com suas colunas e a opção de adicionar uma nova coluna. Ao clicar em uma tabela o formulário será alterado para cadastrar, alterar ou excluir tabelas e quando for clicado em uma coluna o formulário será alterado para cadastrar, alterar

ou excluir colunas.

Para iniciar a criação de tabelas e as classes correspondentes é necessário cadastrar uma nova tabela informando o seu nome que será utilizado para criar a tabela do banco de dados e a respectiva classe em linguagem PHP. Após cadastrar a tabela é necessário cadastrar suas colunas informando os seguintes itens:

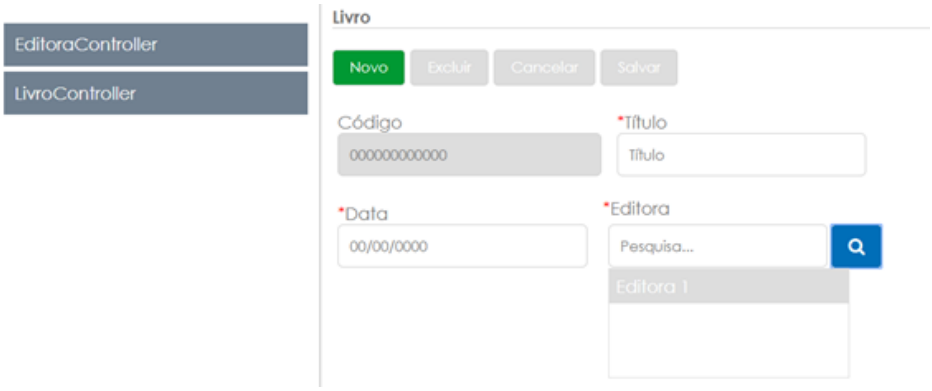
a) Nome: O nome é utilizado para construir a coluna do banco de dados e o atributo do objeto em PHP.
b) Descrição: Utilizado no formulário para o usuário identificar o propósito do campo.
c) Tipo: O tipo pode ser numérico, data, texto entre outros. Utilizado para definir o tipo da coluna do banco de dados e o tipo do atributo PHP. Além dos tipos primários, será disponibilizado um tipo chamado *Tabela* que será utilizado em uma forma de relacionamento entre entidades.

   Por padrão os tipos primitivos serão nomeados com base nos tipos de atributos do PHP, exemplo: *String* para texto, *int* para números inteiros, *DateTime* para datas/horas e *float* para números decimais. No momento em que o *framework* construir as colunas do banco de dados, as nomenclaturas dos tipos de atributos do PHP serão convertidas conforme as nomenclaturas dos tipos da coluna do banco de dados utilizado.
d) Tamanho: Para alguns tipos de colunas do banco de dados é obrigatório informar o tamanho. Por exemplo, para utilizar um campo do tipo texto é necessário informar quantos caracteres a coluna suportará.
e) Somente leitura: Essa opção impede que o usuário digite no campo do formulário. Utilizado para campos que serão preenchidos automaticamente.
f) Obrigatório: Se essa opção estiver habilitada o registro será salvo somente se o campo estiver preenchido, não permitindo valores nulos para a coluna.
g) Chave primária: Define que a coluna será uma chave primária do banco de dados. Poderá ser cadastrado mais de uma coluna para compor uma chave primária.
h) Auto incrementar: Após habilitar essa opção, a cada registro cadastrado o campo da chave primária será preenchido incrementando um valor automaticamente.
i) Máscaras: Utilizado no formulário para evitar que o usuário informe um valor incorreto. Para colunas do tipo data e número o *framework* criará máscaras

automaticamente.

Cada tabela cadastrada contém um botão “Gerar Arquivos” apresentado na Figura 12. Esse botão utiliza os componentes apresentados na seção 4.1.6 para gerar um sistema que permite manipular os dados dessa tabela no banco de dados. A Figura 13 mostra o resultado de um formulário gerado automaticamente após o cadastro de uma entidade denominada “livro”.

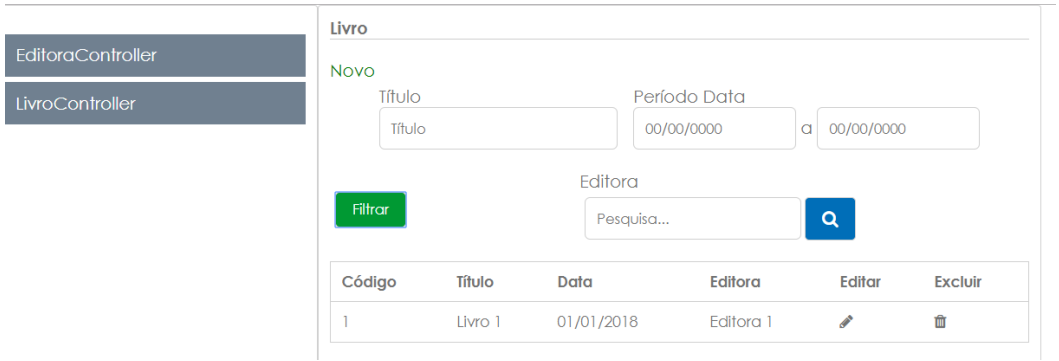
Figura 13 - Formulário para cadastro de livros



Fonte: elaborado pelo autor (2018).

Além de ser construída uma interface contendo formulário para inserir dados, é gerada automaticamente uma tela que permite buscar, editar ou excluir registros cadastrados conforme a figura 14.

Figura 14 - Tabela lista de registros cadastrados



Fonte: elaborado pelo autor (2018).

O *framework* não permite apenas o cadastro de tabelas e atributos primitivos, com este é possível cadastrar relacionamentos de um para muitos ou muitos para muitos entre as tabelas descrito nas seções 4.1.7.1 e 4.1.7.2.

Uma entidade pode relaciona-se com outra entidade de duas formas. A primeira forma ocorre quando um registro de uma entidade precisa conhecer apenas um registro de outra entidade. Por exemplo: uma pessoa possui uma cidade de nascimento. A cidade e a pessoa são entidades distintas, porém a entidade pessoa precisa conhecer a entidade cidade. Essa forma de relacionamento é denominada de relacionamento de um para muitos.

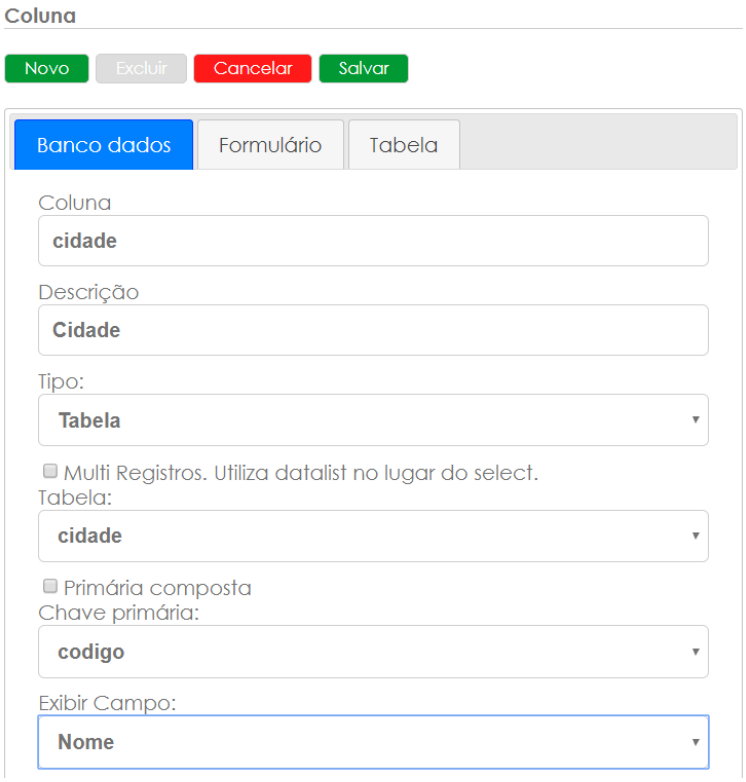
A segunda forma de relacionamento ocorre quando um registro de uma entidade precisa conhecer uma lista de registro de outra entidade. Por exemplo: uma pessoa pode conter uma lista de documentos. Essa forma de relacionamento é denominada de relacionamento de muitos para muitos.

As seções a seguir apresentam a forma no qual o *framework* proposto lida com os relacionamentos.

4.1.7.1 Relacionamento de um para muitos *entre entidades*

Para o exemplo mencionado na seção *4.1.7*, no qual a entidade pessoa precisa conhecer a entidade cidade, a pessoa será considerada uma entidade de destino e a cidade será considerado uma entidade de origem. Para este caso será necessário cadastrar um novo atributo para a entidade de destino e selecionar o tipo Tabela. Conforme ilustra a Figura 15, ao selecionar o tipo tabela será disponibilizada uma lista para escolher uma entidade de origem e um dos seus atributos no qual será utilizado para visualização e pesquisa por usuários, o *framework* buscará automaticamente as chaves primárias.

Figura 15 - Cadastro do relacionamento de um para muitos entre tabelas



Fonte: elaborado pelo autor (2018).

Ao salvar a entidade de destino o *framework* criará a(s) coluna(s) de chave(s) estrangeira(s) na tabela de destino, conforme a(s) coluna(s) primárias da tabela de origem. Se o desenvolvedor gerar as classes em PHP referentes a essa entidade de destino, serão construídos todos os atributos na classe PHP referentes às chaves estrangeiras. Porém, quando o usuário for inserir um novo registro na tela de cadastro, os campos do formulário referentes a essas chaves estrangeiras ficarão invisíveis, sendo possível preencher apenas o campo no qual foi cadastrado o atributo da entidade de destino que gerou as chaves estrangeiras, pois este é o que possuíra a descrição formal do campo.

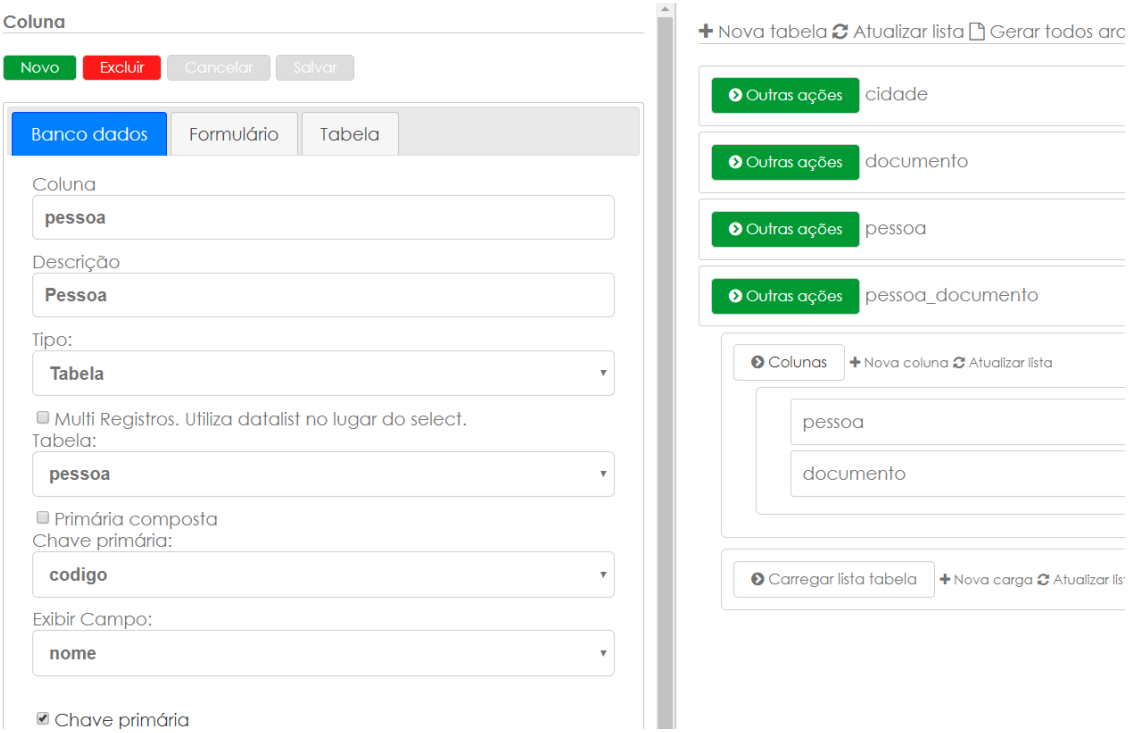
4.1.7.2 Relacionamento de muitos para muitos entre entidades

Em casos no qual um registro de uma entidade de destino precisa conhecer uma lista de registro de uma entidade de origem, será necessário modelar uma nova entidade contendo um atributo chave estrangeira referenciando a tabela de origem e outro atributo chave estrangeira referenciando a tabela de destino. Para cada entidade

será disponibilizada uma opção chamada “Carregar lista de tabela “, no qual será utilizada para vincular a tabela que contém as chaves estrangeiras das entidades de origem e destino. Tanto na entidade de origem quanto na entidade de destino será possível adicionar essa entidade contendo as referências. Após realizar esse vínculo e gerar as classes PHP, o *framework* se encarregará de gerar as telas para incluir ou consultar um registro na lista.

Para o exemplo mencionado na seção *4.1.7*, no qual a entidade pessoa precisa conhecer vários documentos será modelada uma terceira entidade chamada “pessoa_documento” com uma coluna para armazenar uma pessoa contendo uma chave estrangeira com referência para a tabela “pessoa” e outra coluna para guardar o documento contendo uma chave estrangeira referenciando a tabela “documento” conforme a Figura 16. Ambas as colunas formarão uma chave primária.

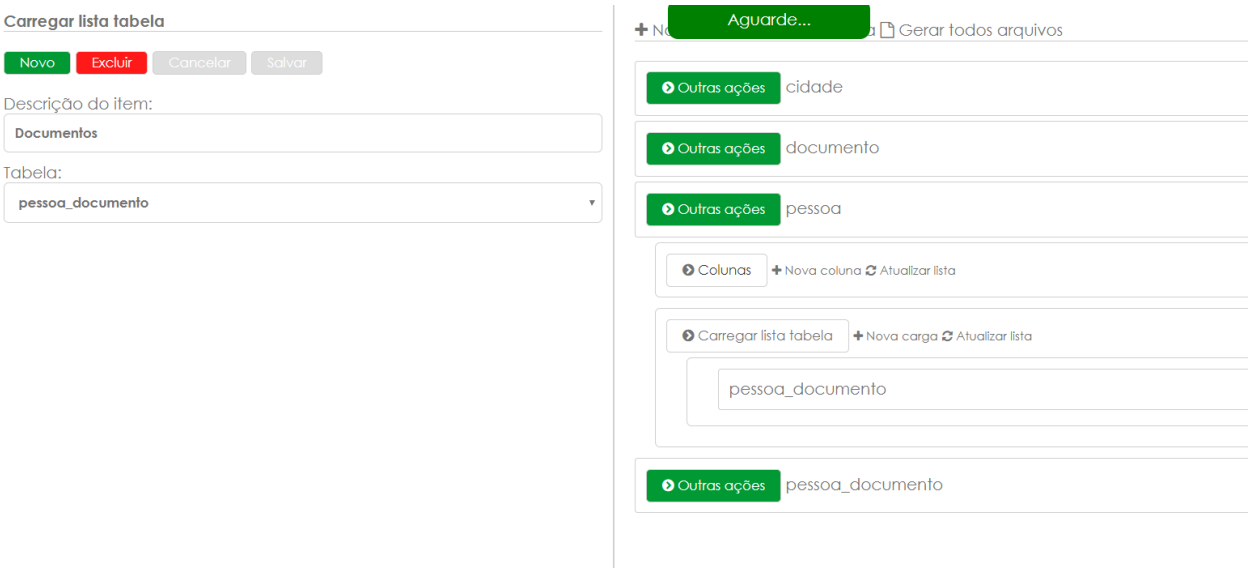
Figura 16 - Cadastro da entidade para o relacionamento de muitos para muitos



Fonte: elaborado pelo autor (2018).

Após cadastrar a terceira tabela é necessário cadastrar o relacionamento de muitos para muitos utilizando a função “Carregar lista de tabela” ilustrado na Figura 17.

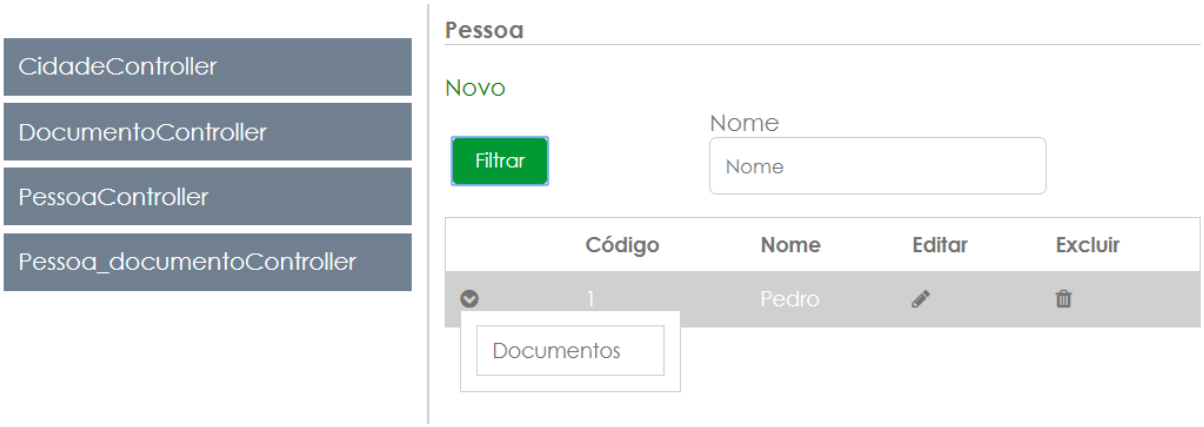
Figura 17 - Cadastro do relacionamento de muitos para muitos



Fonte: elaborado pelo autor (2018).

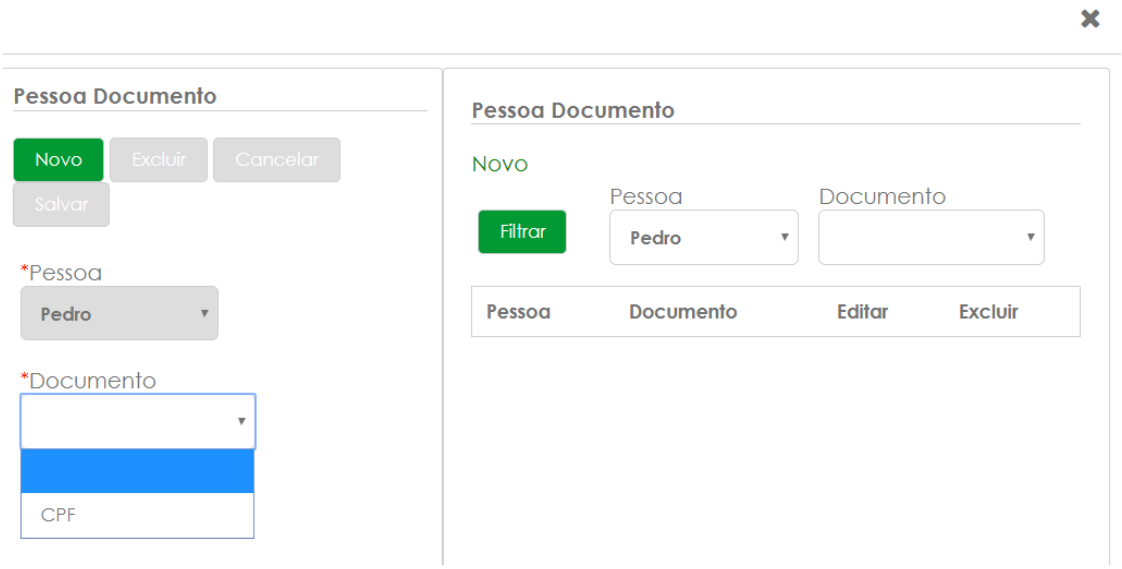
Ao final basta gerar todos os arquivos da tabela que o *framework* se encarrega de gerar a interface no qual uma entidade pode conhecer vários registros de outra entidade. A Figura 18 e a Figura 19 ilustram o resultado final após gerar os arquivos das tabelas “pessoas”, “documento” e “pessoa_documento”.

Figura 18 - Subitem para cadastrar os documentos de uma pessoa



Fonte: elaborado pelo autor (2018).

Figura 19 - Interface para cadastrar vários documentos de uma pessoa



Fonte: elaborado pelo autor (2018).

4.2 Validação da usabilidade

Para avaliar o *framework* foram convidados 7 programadores com conhecimentos em modelagem de banco de dados e desenvolvimento de sistemas. Os convidados realizaram uma tarefa com o objetivo de avaliar a usabilidade da interface oferecida pelos componentes do *framework*. A tarefa consistia em desenvolver um sistema para armazenar dados de clientes. O Apêndice A contém a tarefa que foi aplicada. Após completar a tarefa os convidados responderam o questionário do Anexo A.

4.2.1 Resultados da Validação

Todos os convidados conseguiram realizar a tarefa sem o auxílio de algum manual ou instrução por parte do aplicador da tarefa.

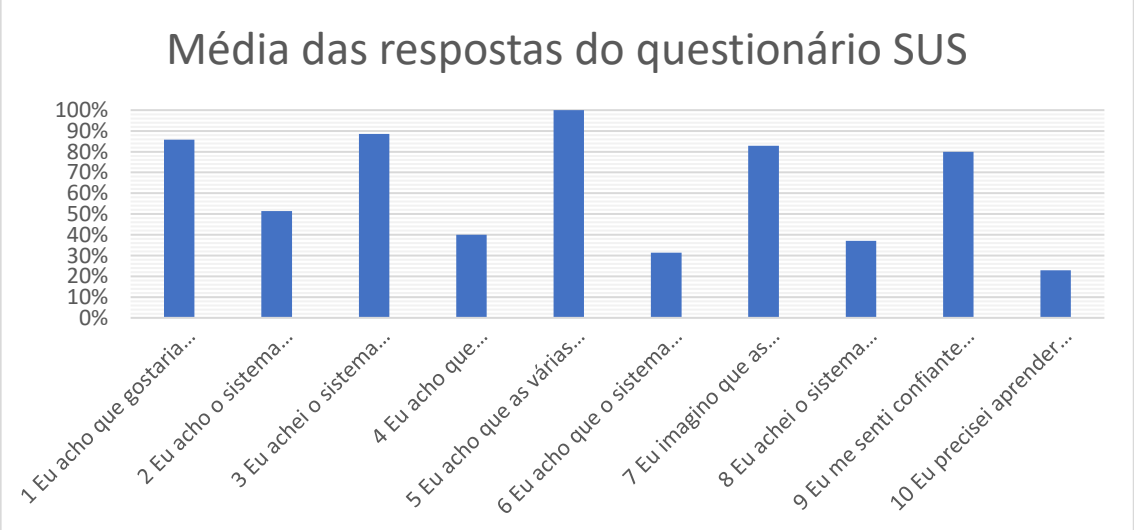
Após a realização dos testes e com base na observação dos convidados constatou-se que a opção para realizar o cadastro das colunas não estava intuitiva. Foram utilizadas as informações fornecidas pelos convidados para a realização de ajustes de melhorias na interface.

Conforme gráfico apresentado na Figura 20 contendo a média das respostas dos convidados, conclui-se:

- 86% dos convidados gostariam de usar o *framework* na elaboração de um sistema para web;
- 51% consideraram a interface complexa;
- 89% consideraram a interface fácil de usar;
- 40% relatarão que precisariam de ajuda para usar o *framework*;
- 100% concordam que todas as funções estão bem integradas;
- 31% informaram uma inconsistência no qual o sistema permitiu salvar uma dada inválida na coluna “Data Nascimento” do cliente;
- 83% relataram que outras pessoas aprenderão facilmente a utilizar os componentes;
- 37% acharam o sistema atrapalhado de usar pelo fato de não ter ficado intuitivo o botão para cadastrar as colunas;
- 80% sentiram-se confiantes em utilizar o *framework*;
- 23% informaram que precisaram aprender outras coisas para conseguir utilizar o *framework*.

Ao final de aplicação da validação aplicou-se o cálculo desenvolvido por John Brooke conforme mencionado na seção 2.6.1 utilizado na medição da usabilidade de uma interface. A pontuação final obtida foi 81,79 e com esse resultado conclui-se que a interface possui uma usabilidade satisfatória conforme *ranking* definido pelo próprio autor John Brooker.

Figura 20 - Média das respostas do questionário SUS



Fonte: elaborado pelo autor (2018).

4.3 Dificuldades

Encontrou-se uma dificuldade no momento de realizar o cadastro de uma nova tabela. Ao salvar a tabela a mesma ainda não tem nenhuma coluna vinculada e o banco de dados bloqueia a inclusão informando que é obrigatório informar uma coluna. Para contornar esse problema, a cada tabela construída pelo *framework* é adicionado uma coluna com o nome *dafault* no qual fica invisível para o usuário.

5 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de um *framework* com componentes utilizados para auxiliar no desenvolvimento de sistemas WEB PHP, automatizando a construção de códigos HTML, JavaScript, PHP e SQL que em conjunto fornecem uma interface completa no qual é possível inserir, alterar, excluir e listar registros em um banco de dados.

Primeiramente, realizou-se um estudo comparando as principais características dos principais *frameworks* disponíveis para PHP com objetivo de construir um novo *framework* contendo as características mais importantes, porém com uma usabilidade intuitiva por meio de interfaces gráficas. Para alcançar os objetivos o desenvolvimento esteve baseado em conceitos de programação orientada a objetos juntamente com os padrões de projetos MVC e DAO.

De acordo com os resultados obtidos ao final do desenvolvimento conclui-se que todos os objetivos foram alcançados conforme projetado no escopo preliminar. A solução permitiu que um grupo de programadores convidados para realizar um teste de validação, desenvolverem um sistema para armazenar dados de clientes. Os convidados responderam o questionário SUS desenvolvido por John Brooke e com os resultados obtidos aplicou-se o cálculo para averiguar o nível de usabilidade, obtendo-se a pontuação de 81,79 o que é considerado satisfatório.

Para os trabalhos futuros, propõe-se adicionar novos componentes com várias funcionalidades, como:

- Gerador de Templates – Facilitar a criação de páginas web;
- Componente para envio de e-mails – Adicionar de forma fácil a funcionalidade de envido de e-mail em uma aplicação;
- Múltiplos BD – Aceitar conexão com outros SGBD, como por exemplo: Oracle *Database* e SQL Server.

REFERÊNCIAS

ALVIM, Paulo. **Tirando o máximo do Java EE 6 Open Source com jCompany Developer Suite**. 3. ed. Belo Horizonte: Powerlogic Publishing, 2010.

ANDRADE, Fernando Francisco. **Desenvolvimento de aplicações web com a utilização dos frameworks Codeigniter e Doctrine.** 2011. 55 f. Trabalho de Conclusão de Curso (Graduação) – Curso Superior de Tecnologia em Desenvolvimento de Sistemas de Informação, Universidade Tecnológica Federal do Paraná, 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/611/1/MD_COADS_2011_1_02.pdf>. Acesso em: 29 out. 2017.

ASSIS, Guilherme Tavares. **Recuperação de falhas:** banco de dados II**.** [201-?]. Disponível em: <http://www.decom.ufop.br/guilherme/BCC441/geral/bd2_recuperacao-de-falhas.pdf>. Acesso em: 29 out. 2017.

BELÉM, Patrick Helder Alvaranga *et al*. **Escolha de um framework para a linguagem de programação PHP através do método AHP clássico.** In: CONGRESSO NACIONAL DE EXCELÊNCIA EM GESTÃO, 10., 2014, Rio de Janeiro, RJ. **Anais...** Rio de Janeiro, 2014. Disponível em: <http://www.mesc.uff.br/publicacoes/cnegpatrickadriano.pdf>. Acesso em: 07 out. 2017.

CYBIS, Walter de Abreu. **Engenharia de Usabilidade: Uma Abordagem Ergômica.** 2003. Disponível em: <http://www.labiutil.inf.ufsc.br/hiperdocumento/unidade3_3_2_1.html>. Acesso em: 16 maio 2018.

CHASE, Nicholas. **Entendendo a Zend Framework, parte 1**: o básico: construindo o leitor perfeito**.** 2006**.** Disponível em: <https://www.ibm.com/developerworks/br/library/os-php-zend1/os-php-zend1-pdf.pdf>. Acesso em: 07 out. 2017.

CUI, Wei *et al*. The research of PHP development framework based on MVC pattern. In: COMPUTER SCIENCES AND CONVERGENCE INFORMATION TECHNOLOGY, 4. 2009., Seul. **Proceedings…** Seul, 2009. p. 947-949. Disponível em: < https://www.computer.org/csdl/proceedings/iccit/2009/3896/00/3896a947.pdf>. Acesso em: 7 out. 2017.

DANNER, Russ. **Getting started with Java-based CMS.** [201-?]. Disponível em: <https://dzone.com/refcardz/getting-started-with-java-based-cms>. Acesso em: 07 out. 2017.

FAYAD, Mohamed; SCHMIDT, Douglas C. Object-oriented application frameworks. **Communications of the ACM**, v. 40, n. 10, p. 32-38, 1997. Disponível em: < https://www.researchgate.net/publication/215617675_Object-Oriented_Application_Frameworks>. Acesso em: 07 out. 2017.

GAMMA, Erich *et al.* **Padrões de projeto:** solução reutilizável de software orientados a objetos. São Paulo: Artmed, 2008.

JOBSTRAIBIZER, F. **Guia profissional PHP**. São Paulo: Digerati Books, 2009.

LAMIM, Jonathan. **MVC:** o padrão de arquitetura de software**.** 2012. Disponível em: <https://www.oficinadanet.com.br/artigo/1687/mvc_-_o_padrao_de_arquitetura_de_software>. Acesso em: 29 out. 2017.

LARMAN, Craig. **Utilizando UML e padrões**. 3. ed. Porto Alegre: Bookman, 2007.

LISBOA, F. G. D. S. **Criando aplicações PHP com Zend e Dojo**. São Paulo: Novatec, 2010.

MELO, Alexandre Altair; NASCIMENTO, Mauricio GF. **PHP profissional:** aprenda a desenvolver sistemas profissionais. São Paulo: Novatec, 2007.

MILANI, André. **MySQL:** guia do programador**.** São Paulo: Novatec, 2006.

MINETTO, Elton Luís. **Frameworks para desenvolvimento em PHP**. São Paulo: Novatec, 2007.

MITTELSTEDT, Vinicius. **UML:** diagrama de atividades. 2012. Disponível em: <http://e-tecinfo.blogspot.com.br/2012/06/uml-diagrama-de-atividades.html>. Acesso em: 29 out. 2017.

MOZILLA. **Sending form data.** 2017. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data>. Acesso em: 06 out. 2017.

NAVATHE, Elmasri. **Sistema de banco de dados.** 6. ed. São Paulo: Pearson, 2010.

NIELSEN, J. **Usability 101:** introduction to usability. 2003. Disponível em: <http://www.useit.com/alertbox/20030825.html>. Acesso em: 16 maio 2018.

ORACLE. **Padrões Core J2EE:** objeto de acesso a dados. [201-?]. Disponível em: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>. Acesso em: 29 out. 2017.

PRESSMAN, S. Roger. **Engenharia de software:** uma abordagem profissional. 8. ed. Porto Alegre: Grupo A Educação, 2016.

RÉ, Reginaldo. **Um processo para construção de Frameworks a partir de engenharia reversa de sistemas de informação baseados na Web:** aplicação ao domínio dos leilões virtuais. 2002. 143 p. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2002. Disponível em:

<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20052003-120738/publico/dissert-f.pdf>. Acesso em: 28 out. 2017.

SANTOS, Alex Lima *et al*. Funcionamento do banco de dados relacional SQL Server 2000 Enterprise. **Revista de Informática Aplicada**, v. 3, n. 1, 2010. Disponível em: < http://www.ria.net.br/index.php/ria/article/view/17/17>. Acesso em: 7 out. 2017.

SAURO, Jeff. **Measuring usability with the system usability scale (SUS)**. 2009. Disponível em: <http://www.measuringusability.com/sus.php>. Acesso em: 16 maio 2018.

SILVA, Patriky Ernany. **Utilização do framework CakePHP para desenvolvimento de websites em PHP**. 2011. 50 p. Trabalho de Conclusão de Curso (Graduação) - Universidade Tecnológica Federal do Paraná, Medianeira, 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/607>. Acesso em: 06 out. 2017.

SILVA, Maurício Samy. **Fundamentos de HTML5 e CSS3**. São Paulo: Novatec, 2015.

TANEMBAUM, S. Andrew; WETHERALL, Davdi. **Redes de computadores**. 5. ed. São Paulo: Pearson, 2011.

TEIXEIRA, Marcelo. **Estudo da utilização de frameworks no desenvolvimento de aplicações web**. 2008. 65 p. Monografia (Graduação) – Bacharelado em Informática, Centro de Ciências Exatas e Tecnológicas, Universidade Estadual do Oeste do Paraná, Cascavel, PR, 2008. Disponível em: <http://www.inf.unioeste.br/~tcc/2008/TCC%20-%20Marcelo%20Teixeira.pdf>. Acesso em: 06 out. 2017.

TEIXEIRA, Daniel *et al*. ManejoSoja3D: ambiente virtual para aprendizado de manejo da cultura da soja. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 28., 2017, Recife, PE. **Anais...** Recife, PE, 2017. Disponível em: < http://brie.org/pub/index.php/sbie/article/view/7606>. Acesso em: 25 jun. 2018.

VENTURA, Plínio. **Entendendo o diagrama de atividades da UML.** 2016. Disponível em: <http://www.ateomomento.com.br/uml-diagrama-de-atividades>. Acesso em: 22 out. 2017.

WINCKLER, Marco Antônio; PIMENTA, Marcelo Soares. Avaliação de usabilidade de sites Web. In: NEDEL, Luciana Porcher (Org.). **Escola de Informática da SBC Sul (ERI 2002)**. Fortaleza: SBC, 2002. v. 1, p. 336- 347. Disponível em: <https://www.irit.fr/~Marco.Winckler/2002-winckler-pimenta-ERI-2002-cap3.pdf>. Acesso em: 06 out. 2017.

APÊNDICE A – Tarefa para teste do Framework

Tarefa: Teste de validação do framework para construção de sistemas Web.

1 DESCRIÇÃO DO FRAMEWORK

Esse framework disponibiliza componentes que facilitam o desenvolvimento de sistemas Web.

2 CENÁRIO

Uma farmácia está lançando uma promoção que realizará o sorteio de um vale medicamentos aos clientes e pediu para você criar um sistema simples para manter o cadastro dos clientes participantes da promoção.

3 TAREFA

Objetivo da tarefa é construir um sistema utilizando apenas interface gráfica para gerenciar o cadastro de clientes. O código fonte desse sistema será gerado automaticamente pelos componentes do framework disponibilizado nas etapas seguintes.

1. Acessar o link do framework http://186.250.184.153/framework/componentes/construtor/
2. Cadastrar uma tabela chamada Cliente.
3. Cadastrar as colunas Nome, Data Nascimento, CPF e telefone.
4. Gerar o arquivo da tabela Cliente.
5. Acessar o link http://186.250.184.153/framework/controladores.php para testar o sistema gerado.
6. Verificar se é possível incluir um novo cliente.
7. Verificar se é possível listar o cliente cadastrado.
8. Verificar se é possível alterar o cliente cadastrado.
9. Verificar se é possível excluir o cliente cadastrado.

Obrigado por participar do teste. Por favor preencha o questionário acessando o link https://www.survio.com/survey/d/X3Q6I4U2Y9H6J1S0C.

ANEXO A – Questionário aplicado