UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

THIARLES FERNANDES DO PRADO

APLICAÇÃO WEB PARA COLETAS DE INFORMAÇÕES DE VISTORIAS IMÓVEIS EMPRESARIAIS E RESIDENCIAIS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

THIARLES FERNANDES DO PRADO

APLICAÇÃO WEB PARA COLETAS DE INFORMAÇÕES DE VISTORIAS IMÓVEIS EMPRESARIAIS E RESIDENCIAIS

Monografia de Conclusão de Curso, apresentada ao Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

APLICAÇÃO WEB PARA COLETAS DE INFORMAÇÕES DE VISTORIAS DE IMÓVEIS EMPRESARIAIS E RESIDENCIAIS

por

THIARLES FERNANDES DO PRADO

Este trabalho de conclusão de curso foi apresentado em 19 de dezembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Beatriz Terezinha Borsoi e Robison Cris Brito. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Vinicius Pegorini
Prof. Orientador (UTFPR)

Beatriz Terezinha Borsoi
Banca (UTFPR)

Robison Cris Brito
Banca (UTFPR)

Robison Cris Brito
Coordenador da IV Especialização
em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

DEDICATÓRIA

Venho de família extremamente humilde, confesso que na minha infância não tinha muitas expectativas de ser ou fazer algo diferente das demais pessoas que estavam na minha volta constantemente. Mas quando eu tinha aproximadamente nove anos, um professor me deu a oportunidade de fazer algo diferente, para tomar um rumo diferente. Então serei eternamente agradecido a todos meus professores que geraram ao longo do tempo algum tipo de oportunidade em minha vida e confiaram na minha capacidade.

AGRADECIMENTOS

Agradeço toda minha família pelo incentivo aos estudos, aos amigos pelas trocas de experiências e principalmente a minha esposa Bruna Jochem do Prado pela paciência e apoio nessa fase de estudos.

Quebrando Barreiras: O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.

José de Alencar

RESUMO

PRADO, Thiarles Fernandes do Prado. Aplicação web para coletas de informações de vistorias de imóveis empresariais e residenciais. 2017. 37f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Diariamente depara-secom vários itens de tecnologia, robôs, *drones*, aplicativos de celulares dos mais diversos tipos e finalidades. Mas, na contramão disso observa-sealgumas funções no mercado de trabalho que pouco evoluíram, pessoas usando prancheta, mapa impresso, soluções que fazem pouco uso da tecnologia e são de baixo rendimento. Uma dessas funções é do inspetor de risco de seguros. Esse profissional visita os imóveis e avalia a viabilidade dos seguros para aprovação ou não da seguradora. Durante a vistoria são coletados dados como instalações elétricas, material da estrutura e cobertura, instalações de gás de cozinha, validade de extintores de incêndio, saídas de emergência e construções vizinhas. Essas informações são registradas por meio de fotografias, que precisam ser enviadas à seguradora juntamente com um laudo em um documento de texto ou planilha de cálculo, que pode variar de acordo com a seguradora. Antes de ser realizada a vistoria, o inspetor recebe os dados do segurado na maior parte das vezes por e-mail, enviado pela empresa intermediadora entre inspetor e seguradora, para após isso ser realizado o agendamento e a vistoria. Este trabalho tem como objetivo a aplicação de uma solução que facilite esse processo. A aplicação desenvolvida permite ao inspetor fazer o acompanhamento de cada pedido de inspeção recebido e atualizar o *status* de cada pedido, pois a empresa intermediadora necessita desses dados atualizados para repassá-los para as seguradoras. O desenvolvimento foi para a plataforma web, pois possibilita o acesso de qualquer local e com os mais variados tipos de dispositivos.

Palavras-chave: Inspetor de risco. Vistoria. Aplicação web.

ABSTRACT

PRADO, Thiarles Fernandes do Prado.Web application for collecting information of surveys of corporate and residential real estate. 2017. 37 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Daily we come across various technology items such as robots, drones, mobile applications of various types and purposes. But on the other hand, we see some functions in the labor market that have not evolved, people using clipboard, printed map, solutions that make little use of technology, and low yield. One of these functions is the insurance risk inspector, where he visits the real estate and assesses the viability of the insurance for approval or not of the insurer. During the survey, data are collected such as electrical installations, structure and cover material, cooking gas facilities, fire extinguisher validity, emergency exits, and neighboring buildings. All this information is recorded through photographs, which need to be sent to the insurer along with a report in a text document or spreadsheet, which may vary according to the insurer. But before the survey is carried out, the inspector receives the insured's data in most of the time by e-mail, sent by the intermediary company between inspector and insurer, after which the scheduling and survey is carried out. This work has as objective the application of a solution that facilitates this process. The developed application allows the inspector to follow up each request for inspection received and update the status of each request, as the intermediary company needs this updated data to pass them on to the insurers. The development was for web platform, because it allows the access of any place and with the most varied types of devices.

Keywords: Inspector of risk. Inspection. Web Application.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

FENASEG	Federação Nacional das Empresas de Seguros Privados
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JDBC	*Java Database Connectivity*
JPA	*Java Persistence*
MVC	*Model View Control*
PIB	Produto Interno Bruto
REST	*Representational State Transfer*
SUSEP	Superintendência de Seguros Privados
UF	Unidade de Federação

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução que é composta pelas considerações iniciais com o escopo e o contexto do trabalho, os objetivos e a justificativa. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

No ramo das seguradoras, o conceito de risco, segundo a FENASEG (1998),"é o evento incerto, ou de data incerta, que independe da vontade das partes, e contra o qual é feito o seguro. O risco é a expectativa de sinistro. Sem risco não pode haver o seguro".

Perim (2002) mostra que a essência do seguro vem de longa data. Os Babilônicos já se utilizavam do seguro para garantir as perdas dos seus camelos durante as grandes travessias no deserto. A prática de seguro utilizada hoje é a evolução das metodologias presentes na antiguidade, o formato da atualidade nasceu no século XIII, na Itália, graças ao desenvolvimento das negociações econômicas no Mediterrâneo. Ela se aprimora no meio do século XVIII, sob o impulso dos mercadores e armadores britânicos, conforme registram Ruffat, Caloni e Laguerre (1990).

O sistema de seguros brasileiro surgiu em 1808 com a abertura dos portos brasileiros ao comércio de outros países. Desde o seu surgimento, os contratos de seguro evoluíram, mas o objetivo continua sendo o mesmo, indenizar o segurado por eventuais prejuízos que venha a ter.

Baseando-se na experiência pessoal do autor deste trabalho que atuou fazendo vistorias para uma empresa que faz a intermediação entre as seguradoras, as corretoras, os segurados e os inspetores, é apresentada a metodologia de trabalho de um inspetor de risco. No formato tradicional, o profissional recebe um arquivo, normalmente por *e-mail*, com dados pessoais e endereço do segurado. Esses dados são anotados em uma agenda e em seguida é realizada ligação para o cliente agendando a visita. Em um segundo momento, é realizada a vistoria, preenchendo um questionário pré-impresso com dados do local. Além disso, são obtidas fotografias internas e externas do ambiente. Em uma última etapa é realizado o preenchimento de um lauto técnico, normalmente via editor de texto ou planilha de cálculo,

repassando a limpo todas as informações coletadas, dentro de um padrão da seguradora e retornado o *e-mail* com o anexo de fotos e laudos.

A aplicação proposta visa melhorar esse processo, evitando, principalmente, a duplicidade de trabalho e agilizar o processo por meio de uma aplicação *web*, que possa ser utilizada pelo inspetor de risco e, também, outros atores envolvidos no processo.

É fundamental que a aplicação seja *web*, pois o inspetor passa o dia todo em campo, e necessita fazer os lançamentos de qualquer local. E nem sempre ele tem em mãos equipamentos com grande capacidade de processamento, então se faz necessário uma ferramenta de fácil utilização.

O objetivo geral deste trabalho é desenvolver um software para inspetores de seguradoras. E os objetivos específicos do trabalho são:

- Permitir a consulta de informações dos pedidos de vistoria;
- Agilizar o processo de atualização do *status* das solicitações;
- Permitir anexar os documentos na vistoria;
- Manter histórico de vistorias.

1.2 JUSTIFICATIVA

O Brasil é líder da América Latina em prêmios gerados, isso representa uma grande participação no Produto Interno Bruto(PIB), saltando de um nível de 2,59% em 2003 para o patamar de 3,82% em 2016, o dobro do registrado na década de 90 (SUSEP, 2017).

Do século XIX ao XXI, o mercado de seguros só cresceu, movimentando mais de R$ 210 bilhões por ano no País, de acordo com a Federação Nacional das Empresas de Seguros Privados (FENASEG, 2017).

Conforme dados apresentados pela Superintendência de Seguros Privados (SUSEP, 2017), o segmento de seguro patrimonial representa a maior parte dos acionamentos de seguro nos últimos anos, como é apresentado na Figura 1.

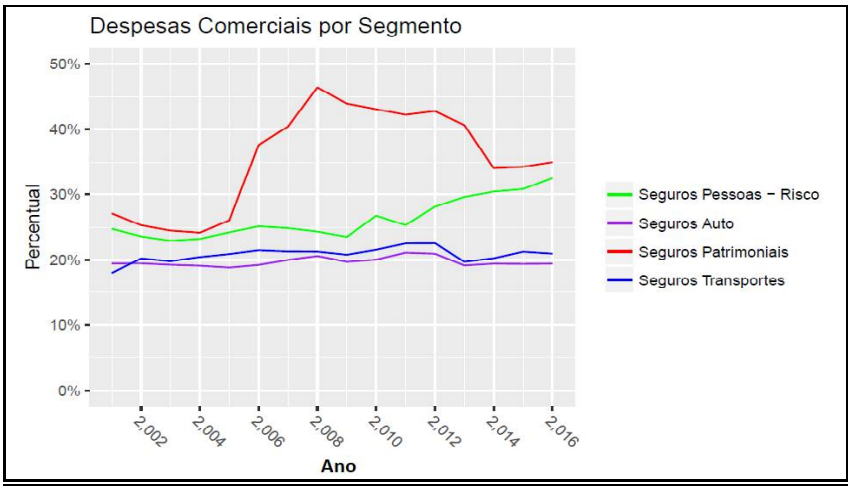


Figura 1 – Dados sobre segmentos de seguro
Fonte: SUSEP (2016, p.20).

Devido ao expressivo número de pessoas que esse setor envolve, identificou-se a necessidade de uma aplicação *web* para agilizar o trabalho que elas realizam. Focando na função do inspetor de seguro, foi projetada uma aplicação que permite o inspetor realizar o acompanhamento das solicitações de vistoria, fazer *download* e verificar as informações sobre o segurado em que será realizada a vistoria, além de permitir mobilidade no envio e consulta dessas informações, pois a aplicação será desenvolvida para plataforma *web*.

2 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. Também é apresentada a sequência das atividades desenvolvidas para a realização do trabalho.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas para modelar e implementar o sistema.

Ferramenta/ Tecnologia	**Versão**	**Referência**	**Finalidade**
Linguagem Java	JDK 1.8	http://www.oracle.com	Linguagem de programação.
Postgres	7.0.5	https://limonte.github.io/sweetalert2/	Componente JavaScript para criação de alertas.
Hibernate	5.2.11	http://www.hibernate.org/	Efetuar o mapeamento objeto-relacional e a persistência dos dados.
Spring Boot	1.5.9	https://projects.spring.io/spring-boot/	Facilita a configuração e publicação da aplicação
Spring Data	1.4.3	https://projects.spring.io/spring-data/	*Framework* para persistência dos dados.
Spring Security	5.0.1	https://projects.spring.io/spring-security/	Estrutura de autenticação e controle de acesso.
Maven	2017-12-10	https://maven.apache.org/	Gerenciador de dependências.
Thymeleaf	3.0.9	http://www.thymeleaf.org/	Modelo para desenvolvimento de aplicações *web*.
Bootstrap	3.3.7	https://getbootstrap.com/	*Framework* de interface gráfica.

Quadro 1 – Ferramentas e tecnologias a serem utilizadas

Na sequencia serão detalhadas as principais ferramentas utilizadas no desenvolvimento.

2.1.1 SPRING FRAMEWORK

Afonso (2017) define o Spring *framework*como um conjunto de projetos que visam facilitar várias etapas do desenvolvimento de software, de maneira simples e flexível. Ele disponibiliza cobertura em muitas áreas, como segurança no acesso ao banco de dados com o Spring Data, segurança com o Spring Security, dentre outros. E, além disso, os seus projetos são de código aberto.

Dentre as suas principais características que são apontadas pelos desenvolvedores no site oficial do Spring, destacam-se:

- Spring MVC (*Model View Control* – MVC) - é um *framework* que auxilia na separação de responsabilidades nos papéis de tratamento das requisições. A camada *Model* é responsável pelas consultas no banco de dados. É nela que serão utilizados o *Java Persistence Application* (JPA) e o Hibernate. A *View* renderiza e transforma em *HyperText Markup Language*(HTML) o resultado obtido na consulta ao banco de dados, para que o usuário consiga visualizar e compreender a informação. A camada *Controller* recebe as requisições realizadas pelo usuário e as repassa para as demais camadas.
- Suporte para *Java Database Connectivity*(JDBC) e JPA - visa simplificar o acesso das tecnologias ao banco de dados. Possui vários projetosinternos, como: Spring Data JPA, Spring Data Key Value, Spring Data *Representational State Transfer*(REST), entre outros.
- Injeção de dependências (*Dependency Injection* - DI) - é um tipo de inversão de controle, que nomeia o processo de prover instâncias de classes. Tem como vantagem o baixo acoplamento entre classes de um mesmo projeto.

2.1.2 MAVEN

O Maven é uma ferramenta de gerenciamento, construção e implantação de projetos, facilita processo de gerenciamento de dependências e do*build*.A unidade básica de configuração do Maven é um arquivo chamado pom.xml, que deve ficar na raiz do projeto. Ele é um arquivo conhecido como *Project Object Model* e nele é declarada a estrutura, as dependências e as características do projeto (MAVEN,2017).

2.1.3 THYMELEAF

O Thymeleaf é um modelo para desenvolvimento de aplicações *web* Java no lado servidor. É um software de código aberto e seu principal objetivo é prover uma forma bem organizada para a geração das páginas. É uma biblioteca criada para auxiliar na criação da camada *view* e oferece ótima integração com Spring MVC (TIMPE,2017).

2.1.4 POSTGRESQL

O Postgres é um banco de dados de código aberto, com estrutura relacional e é compatível com vários sistemas operacionais. Possui recursos como replicação assíncrona, controle de concorrência, *backups* e otimizador de consultas (POSTGRESQL,2017).

3 RESULTADOS

A seguir é apresentado o fluxo de trabalho, as principais telas da aplicação e as suas funcionalidades. Também serão apresentadas partes do código desenvolvido.

3.1 ESCOPO DO SISTEMA

A aplicação web desenvolvida tem por objetivo facilitar o processo de vistorias de seguros, evitando a duplicidade de lançamentos e agilizar a busca de informações. As telas de cadastro são: pessoas, cidades, estados, região e tipos de vistorias.

Toda a parte operacional se baseia na tela de pedido, a partir dela, o usuário poderá executar vários processos, acompanhar *status* de andamento e prazos, além de anexar e fazer *download* dos arquivos que envolvem cada vistoria.

O inspetor terá aprincipal tela na aplicação com as inspeções e as situações atualizadas, podendo atualizar a qualquer momento, também acompanhando prazos, e no final do ciclo poder enviar dados e documento. Ficando, assim, encerrada areferida vistoria, porém mantendo histórico.

3.2 MODELAGEM DO SISTEMA

A seguir, são apresentados os requisitos funcionais da aplicação.

F01 - Cadastrar Cidade;

F02 - Cadastrar Estado;

F03 - Cadastrar País;

F04 - Cadastro de Pessoa Física/Jurídica;

F08 - Consulta Pedidos Cadastrados;

F09 - Anexar Formulários Preenchidos;

A Figura 2 apresenta o fluxo de trabalho do inspetor. O fluxo inicia com a solicitação de seguro feita pelo cliente à corretora. Após isso, é feita a identificação do tipo do seguro e é encaminhada uma solicitação à seguradora. A solicitação passa por uma análise e é repassada

para o inspetor. O inspetor recebe a solicitação, entra em contato com o cliente e realiza o agendamento. Em seguida, é realizada a vistoria, são preenchidos os laudos e eles são enviados para a seguradora.

A etapa que foi desenvolvida nessa aplicação é a do inspetor. Ele pode consultar os dados da vistoria, realizar os agendamentos e anexar os arquivos dos laudos realizados.

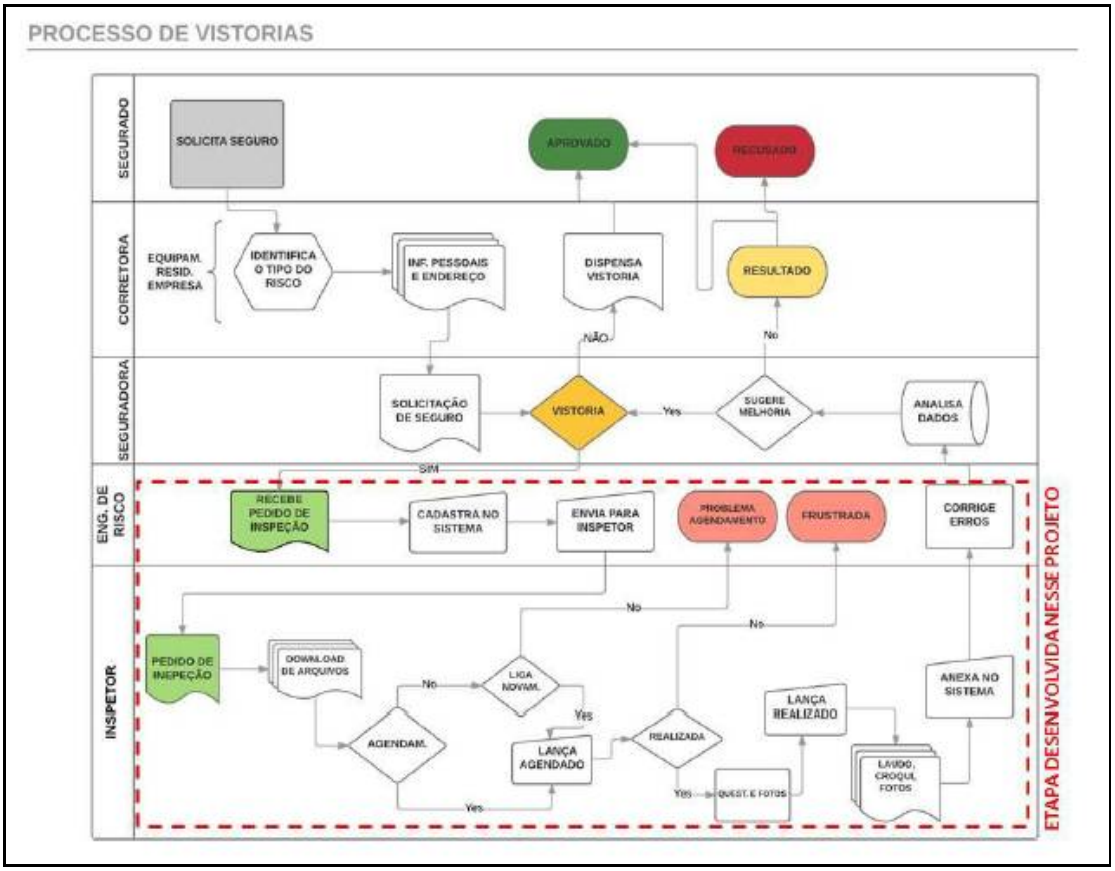


Figura 2 - Fluxo de Trabalho

Na Figura 3 são apresentados os casos de uso da aplicação. Os usuários definidos como administradores utilizarão todos os recursos disponíveis na aplicação, usuários do tipo inspetores utilizam recursos de vistoria.

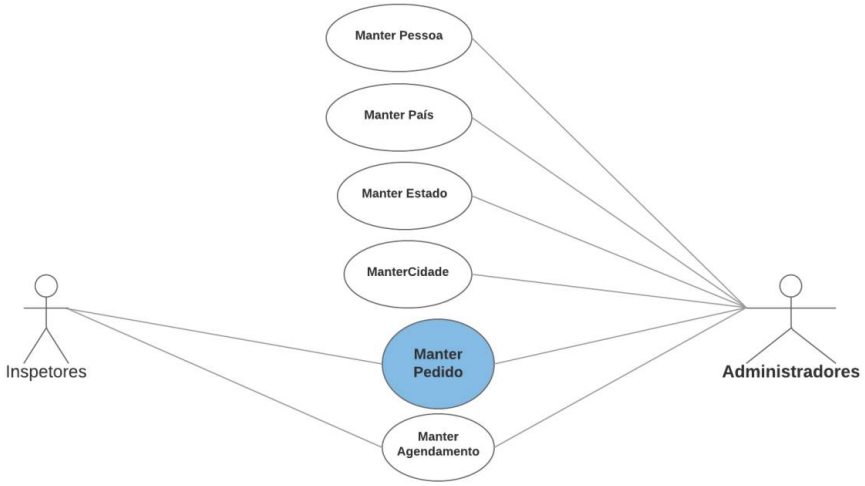


Figura 3 – Diagrama de casos de uso

Na Figura 4 é apresentado o diagrama de classes, esse diagrama é uma representação da estrutura das classes que servem de modelo para os objetos utilizados na aplicação. Cada classe do diagrama representa uma tabela do banco de dados com seus devidos campos e tipos. A principal classe do sistema é “Pedido” que é responsável por cadastrar todos os dados do pedido, estes dados serão necessários para o inspetor agendar a vistoria que está representada pela classe “AgendaPedido”. A classe “PedidoAnexo” é responsável por gravar no banco de dados o número e nome do arquivo anexado no pedido.

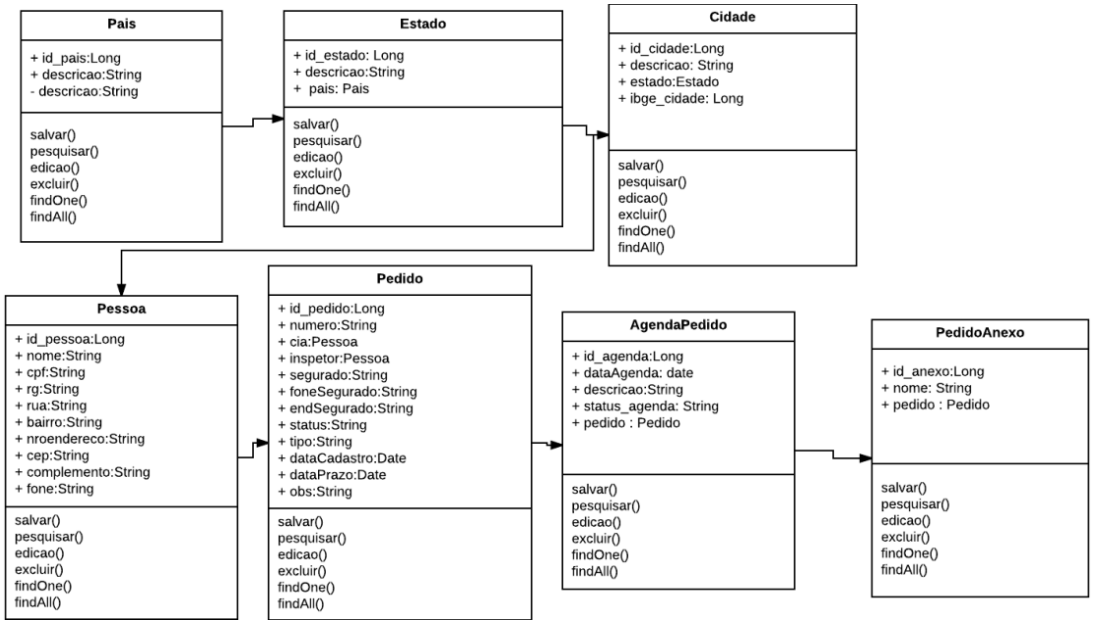


Figura 4 – Diagrama de Classes

3.3 APRESENTAÇÃO DO SISTEMA

Neste subcapítulo são apresentadas as principais telas e funcionalidades da aplicação. Para acessar o sistema o usuário precisa estar autenticado. A tela de autenticação, a primeira que é apresentada ao usuário ao acessar o sistema, é exibida na Figura 5. O usuário deverá informar o seu *login* e senha e clicar no botão “Entrar” e, se desejar, pode marcar a opção “Lembrar” para que não precise informar seus dados no próximo acesso. Após o *login*, o usuário é direcionado para a tela inicial, que é exibidana Figura 6.

Figura 5–Tela de login

Na tela inicial do sistema são listadas as solicitações de vistoria vinculadas ao usuário, no caso, o inspetor. São listadas informações como: o nome do segurado, telefone, endereço, o prazo em que a vistoria deve ser feita e o seu *status*. É possível incluir novos pedidos, editar os já cadastrados e excluí-los.

Na parte superior da tela mostrada na Figura 6 está disponível o menu do sistema, no qual são listadas as opções de cadastro, pedidos e a opção de sair.

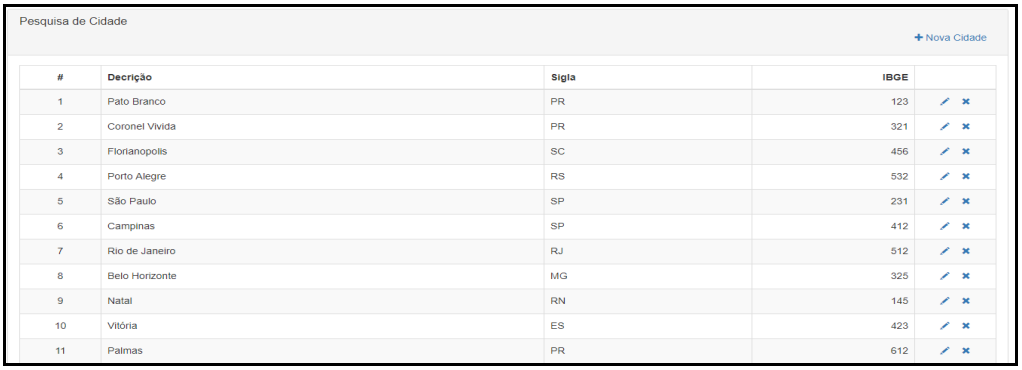
Figura 6 - Tela de solicitações de vistorias e menu de cadastros

Todos os cadastros do sistema são compostos por uma tela de listagem e uma tela de cadastro. Na Figura 7 é apresentada a tela de cadastro de cidades. Para o cadastro de cidades são necessários informar a descrição da cidade, o código no IBGE e deve ser selecionada a

Unidade de Federação (UF). Nessa tela, é possível realizar a inclusão ou alteração de registros. O campo de UF é no formato de Combo, sendo carregada uma lista de UFs e o usuário deve selecionar o registro desejado.

Figura 7 - Cadastro de cidade

Na Figura 8 é apresentada a pesquisa de cidades, com informações que estão salvas no banco de dados, os principais campos são: descrição, sigla do estado e código do IBGE. Por padrão das telas de pesquisa, no canto superior direito trazem uma opção para incluir um novo registro e o final de cada linha possui botões para excluir ou editar o registro.



#	Decrição	Sigla	IBGE
1	Pato Branco	PR	123
2	Coronel Vivida	PR	321
3	Florianopolis	SC	456
4	Porto Alegre	RS	532
5	São Paulo	SP	231
6	Campinas	SP	412
7	Rio de Janeiro	RJ	512
8	Belo Horizonte	MG	325
9	Natal	RN	145
10	Vitória	ES	423
11	Palmas	PR	612

Figura 8 - Listagem de Cidade

Na Figura 9 é apresentada a tela de cadastro de pessoas. Esse cadastro é utilizado para cadastrar inspetores e seguradoras. Os principais campos são nome e endereço, o campo cidade carrega uma listagem de todas as cidades cadastradas no sistema. Esse cadastro não vale para segurado, pois nem sempre é o titular quem recebe o inspetor.

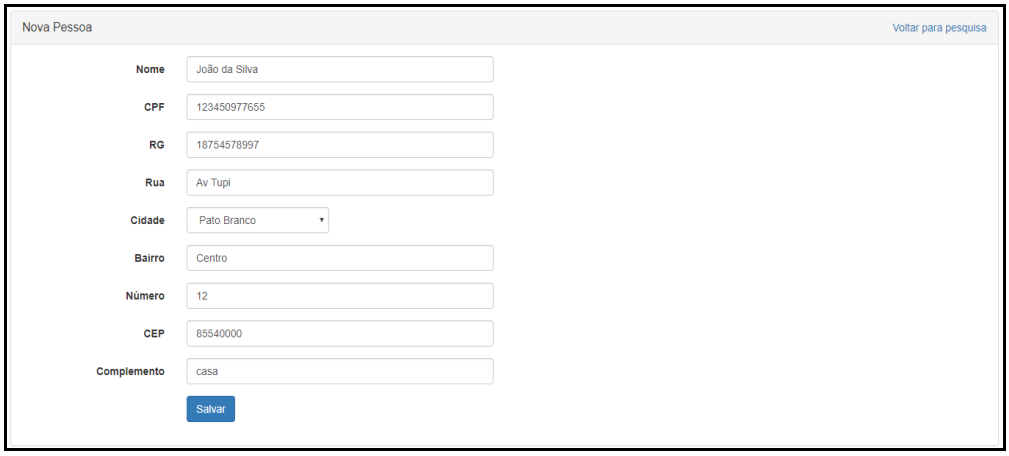
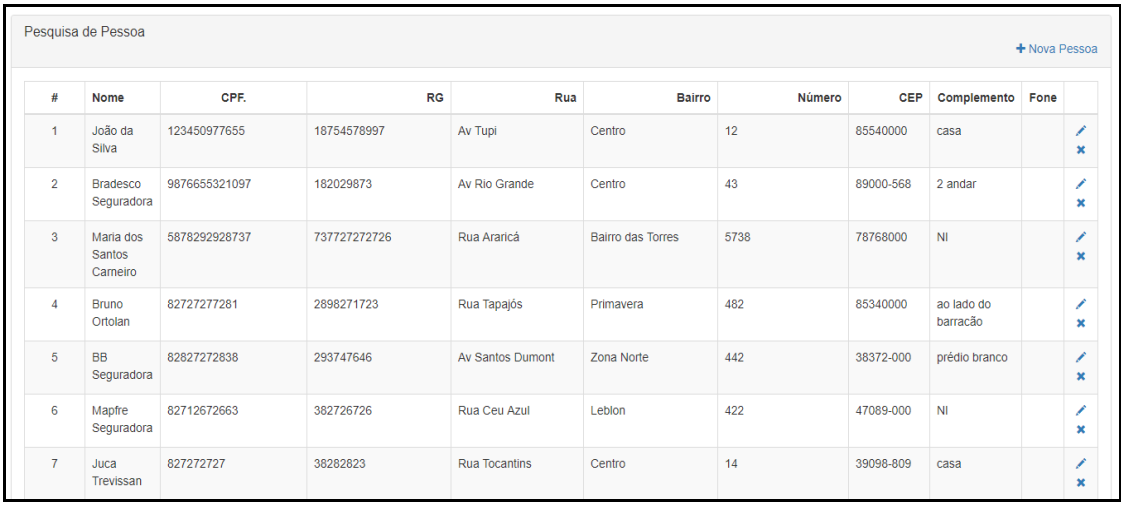


Figura 9 - Cadastro de pessoa

Na Figura 10 é apresentada a pesquisa de pessoas, com informações que estão salvas no banco de dados na tabela “pessoa”. Seguindo o padrão das demais telas de pesquisa, essa tela traz no canto superior direito uma opção para incluir uma nova pessoa, também ao final de cada linha possui botões para excluir ou editar o cadastro.



#	Nome	CPF.	RG	Rua	Bairro	Número	CEP	Complemento	Fone
1	João da Silva	123450977655	18754578997	Av Tupi	Centro	12	85540000	casa	
2	Bradesco Seguradora	9876655321097	182029873	Av Rio Grande	Centro	43	89000-568	2 andar	
3	Maria dos Santos Carneiro	5878292928737	737727272726	Rua Araricá	Bairro das Torres	5738	78768000	NI	
4	Bruno Ortolan	82727277281	2898271723	Rua Tapajós	Primavera	482	85340000	ao lado do barracão	
5	BB Seguradora	82827272838	293747646	Av Santos Dumont	Zona Norte	442	38372-000	prédio branco	
6	Mapfre Seguradora	82712672663	382726726	Rua Ceu Azul	Leblon	422	47089-000	NI	
7	Juca Trevissan	827272727	38282823	Rua Tocantins	Centro	14	39098-809	casa	

Figura 10 - Listagem de Pessoa

Na Figura 11 é apresentada uma das principais telas do sistema, nela inicia-se o processo de vistoria, ao receber a solicitação de vistoria da seguradora, o usuário “inspetor ou administrador” cadastra um novo pedido, informando a seguradora, os anexos e o prazo indicado para a realização da vistoria. O campo “nome segurado” é do tipo String, para permitir informar a pessoa que receberá e acompanhará o inspetor durante a vistoria, nesse caso pode ser o próprio inspetor ou algum familiar. O campo “status” deve ser atualizado

conforme será mudado de fase do pedido: cadastro, enviado ao inspetor, agendado, problema no agendamento, realizada ou cancelada.

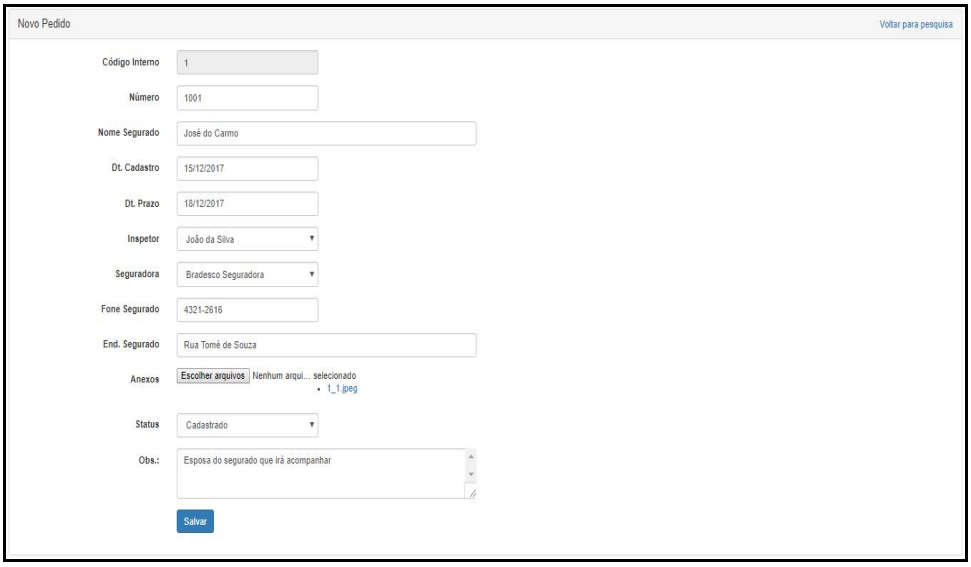


Figura 11–Cadastro de pedido

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para a implementação do sistema foi utilizada a linguagem Java EE, o *framework* Spring, a *Integrated Development Environment*(IDE) Eclipse Luna, com a persistência de dados no Postgres. Ao iniciar o sistema a primeira tela é a de autenticação, nela, obrigatoriamente, o usuário precisa informar seus dados de acesso para navegar pelas demais telas, nessa rotina aplica-se o *framework* Spring Security.

Na Listagem 1 está o código que faz a validação de usuário, que deve estar autenticado no sistema para navegar nas páginas que estão disponíveis no menu fixado ao topo da página. Pode-se observar o uso do Spring Security com a anotação "@EnableWebSecurity", que importa as dependências necessárias. No método configure() são realizadas as configurações de segurança que limitam acessos por tipos de usuários. O comando "loginPage" define que a URL "/entrar" é a principal.

```
package br.edu.utfpr.pb.vistoria.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
```

```
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import br.edu.utfpr.pb.vistoria.repository.UsuarioRepository;
import br.edu.utfpr.pb.vistoria.service.UsuarioService;

@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{

	@Autowired
	private UsuarioRepository usuarioRepository;
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.csrf().disable()
			.exceptionHandling().accessDeniedPage("/erro403")
			.and().formLogin().loginPage("/entrar")
			.defaultSuccessUrl("/pedido/")
			.failureUrl("/login?error=bad_credentials").permitAll()
			.and().authorizeRequests()
				.antMatchers("/pedido/**").permitAll()
				.antMatchers("/pedido/**").hasAnyRole("USER","ADMIN")
				.antMatchers("/cidade/**").hasRole("ADMIN" )
				.antMatchers("/estado/**").hasRole("ADMIN")
				.antMatchers("/pais/**").hasRole("ADMIN")
				.antMatchers("/usuario/**").hasRole("ADMIN")
				.antMatchers("/pessoa/**").hasRole("ADMIN")
				.antMatchers("/**").hasRole("USER");
	}

	@Override
	public void configure(WebSecurity web) throws Exception {
		web.ignoring().antMatchers("/webjars/**");
		web.ignoring().antMatchers("/assets/**");
	}

	@Bean
	public UserDetailsService userDetailsService() {
		return new UsuarioService(usuarioRepository);
	}
	@Bean
	public PasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder(10);
	}

	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
	}
}
```

Listagem 1 – Configurações de segurança

Na Listagem 2 consta parte de código que faz o mapeamento objeto relacional (ORM) da tabela “pedido”. Na API JPA do Java a anotação “@Entrity” define que a classe é

mapeada e será persistida no banco de dados, além de poder definir um nome específico para cada tabela e cada coluna no banco de dados.

Java pode persistir em diversos bancos de dados, nesse caso foi utilizado o Postgres.Basta configurar nas propriedades a conexão com o banco de dados utilizado. Essa padronização evita que seja necessário alteração do código fonte caso seja utilizado outro banco de dados. O arquivo de configuração da conexão do banco de dados pode ser observado na Listagem 3.

```
…
@Entity(name="pedido")
public class Pedido {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	public Long id_pedido;

...
```

Listagem 2 – Persistência de dados

Na Listagem 3 estão as configurações de conexão com o banco de dados, a partir dessas configurações é que Java interage com banco de dados para fazer as operações de inclusão, exclusão, atualização e consulta aos dados. Por meio da URL presente em: “spring.datasource.url=jdbc:postgresql://localhost:5432/vistoria1” é realizada à conexão ao banco de dados com nome “vistoria1”, na sequência devem estar usuário e senha definidos ao criar o banco de dados.

```
server.port = 8090
spring.mvc.view.prefix: /WEB-INF/jsp/
spring.mvc.view.suffix: .jsp
spring.jpa.hibernate.ddl-auto:update
spring.jpa.show-sql: true
spring.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.datasource.url=jdbc:postgresql://localhost:5432/vistoria1
spring.datasource.username=postgres
spring.datasource.password=admin
spring.datasource.driverClassName=org.postgresql.Driver
```

Listagem 3 – Configurações de conexão com banco de dados

Na Listagem 4é exibido um trecho do código da classe “PedidoController”. Esse método faz com que o usuário seja direcionado a uma nova página no momento que o usuário clicar na opção “Novo Pedido”. A anotação “@RequestMapping” define a URL que o método novo irá tratar quando a camada de visão da aplicação enviar requisições do tipo HTTP GET. Ainda na Listagem 4 está o retorno do método consta “ModelAndView”, ele define qual

página será exibida, além de adicionar uma lista de pessoas que será exibida nos campos “seguradora” e “inspetor”, carregando o nome gravado na tabela “pessoa”.

```
…
@RequestMapping("/novo")
	public ModelAndView novo() {
		ModelAndView mv = new ModelAndView(CADASTRO_VIEW);
		mv.addObject(new Pedido());

		mv.addObject("todasPessoas", pessoas.findAll());
		return mv;
	}
...
```

Listagem 4 - Mapeamento para inserir novo pedido

Na Listagem 5é exibido um trecho do código da classe “PedidoController”. O método “salvar” cria um novo registro de pedido no banco de dados. Além dos campos do tipo String, Date, o método requer como parâmetro o anexo do pedido do tipo “MultipartFile”. Ao anexar um arquivo, esse trecho de código grava na tabela “pedido_anexo” o ID e Nome do arquivo e caminho onde foi salvo, facilitando assim a edição e entendimento de qual arquivo foi anexado.

```
…
@RequestMapping(method = RequestMethod.POST)
	public String salvar(@Validated Pedido pedido, Errors errors, RedirectAttributes attributes,
			@RequestParam("anexos") MultipartFile[] anexos, HttpServletRequest request) {

		if (errors.hasErrors()) {
			return CADASTRO_VIEW;
		}
		try {
			pedidos.save(pedido);
			attributes.addFlashAttribute("mensagem", "Pedido salvo com sucesso!");

			if (anexos.length > 0){
				File dir = new File(request.getServletContext().getRealPath("/anexos/" +
pedido.getId_pedido()));
				if (!dir.exists()) {
					dir.mkdirs();
				}
				String caminhoAnexo = request.getServletContext().getRealPath("/anexos/"
+ pedido.getId_pedido() + "/");
				String extensao = "";
				String nomeArquivo = "";
				int i = 1;
				List<PedidoAnexo> lPA =
pedidosAnexo.findByPedidoIdPedido(pedido.getId_pedido());
				if (lPA != null){
					i = lPA.size() + 1;
				}
				PedidoAnexo pa;
				for (MultipartFile a : anexos) {
					extensao =
a.getOriginalFilename().substring(a.getOriginalFilename().lastIndexOf("."),
```

```
                                                    a.getOriginalFilename().length());
                                        nomeArquivo = pedido.getId_pedido() + "_" + i + extensao;
                                        salvarAnexoDisco(caminhoAnexo + nomeArquivo, a.getBytes(),
request);

                                        pa = new PedidoAnexo();
                                        pa.setNome(nomeArquivo);
                                        pa.setPedido(pedido);
                                        pedidosAnexo.save(pa);
                                        i++;
                                }
                        }
                        return "redirect:/pedido";
                } catch (DataIntegrityViolationException e) {
                        errors.rejectValue("a", null, "Formato de data inválido");
                        return CADASTRO_VIEW;
                } catch (IOException e) {
                        e.printStackTrace();
                        errors.rejectValue("b", null, "Erro no anexo");
                        return CADASTRO_VIEW;
                } catch (Exception e) {
                        e.printStackTrace();
                        errors.rejectValue("c", null, "Erro no anexo");
                        return CADASTRO_VIEW;
                }
        }
...
```

Listagem 5 - Salvar anexos na tela de pedidos.

Na Listagem 6 está o código HTML que faz a interface visual que aparece para o usuário. Neste trecho de código é possível observar o uso do Thymeleaf utilizando a importação do “LayoutPadrao”. Ao criar um leiaute no Thymeleaf evita-se aduplicidade do código, facilitando a manutenção. Caso seja alterado algo no leiaute padrão, todas as páginas que fazem seu uso receberão automaticamente as novas alterações. Na *tag* “<tr th:each="pedido : ${pedidos}">”, o código “th:each” é responsável por percorrer cada um dos pedidos e lista-los na tabela, em que ${pedidos} é a lista de pedidos vinda do servidor.

```
<!DOCTYPE html>
<html
xmlns="http://www.w3.org/1999/xhtml"xmlns:th="http://www.thymeleaf.org"xmlns:layout="http://www.ultraq.
net.nz/thymeleaf/layout"layout:decorator="LayoutPadrao">
<head>
<title>Pesquisa de Pedidos</title>
</head>
<section layout:fragment="conteudo">
        <div class="panel panel-default">
                <div class="panel-heading">
                        <div class="clearfix">
                                <h1 class="panel-title">Pesquisa de Pedidos</h1>
<a class="btn btn-link aw-link-panel" href="/pedido/novo"><i class="fa fa-plus"></i> Novo Pedido</a>
                        </div>
                </div>
                <div class="panel-body">
```

```
                        <table class="table table-bordered table-striped">
                                <thead>
                                        <tr>
                                                <th class="text-center col-md-1">#</th>
                                                <th class="text-center col-md-1">NÚMERO</th>
                                                <th>CIA</th>
                                                <th>SEGURADO</th>
                                                <th>INSPETOR</th>
                                                <th class="text-center col-md-2">FONE</th>
                                                <th>CADASTRO</th>
                                                <th>PRAZO</th>
                                                <th>ENDEREÇO</th>
                                                <th class="text-center col-md-1">STATUS</th>
                                                <th class="col-md-1"></th>
                                        </tr>
                                </thead>
                                <tbody>
                                        <tr th:each="pedido : ${pedidos}">
                                        <td class="text-center" th:text="${pedido.id_pedido}">1</td>
                                        <td th:text="${pedido.numero}"></td>
                                        <td class="text-right" th:text="${{pedido.cia.nome}}"></td>
                                        <td class="text-right" th:text="${{pedido.segurado}}"></td>
                                        <td class="text-center" th:text="${{pedido.inspetor.nome}}"></td>
                                        <td th:text="${pedido.foneSegurado}"></td>
                                        <td th:text="${pedido.dataCadastro}"></td>
                                        <td th:text="${pedido.dataPrazo}"></td>
                                        <td th:text="${pedido.endSegurado}"></td>
                                        <td th:text="${pedido.status}"></td>
                                        <td class="text-center">
...
```

Listagem 6 – Importando leiaute padrão com Thymeleaf

Na Listagem 7 está parte do código HTML da tela que faz o cadastro de um novo pedido. O trecho cria o campo “Código Interno” na interface que aparece para o usuário. No inicio do código está a *tag* “form” que define que parte do código é um formulário e com parâmetro *action* que contém a URL que será chamada no *controller* de “pedido”. Ainda na Listagem 7, as *tags* “input” servem para entrada de valores do usuário, ao clicar no botão salvar será realizada uma requisição ao *controller*para a persistência dos dados no banco.

```
…
<form class="form-horizontal" method="POST" th:action="@{/pedido}" th:object="${pedido}"
enctype="multipart/form-data">
        <div layout:include="Mensagem"></div>
        <div class="panel panel-default">
                <div class="panel-heading">
                <div class="clearfix">
                <h1 class="panel-title aw-titulo-panel">Novo Pedido</h1>
                <a class="btn btn-link aw-link-panel" href="/pedido">Voltar para pesquisa</a>
                </div>
                </div>
                <div class="panel-body">
                <input type="hidden" th:field="*{id_pedido}"/>
                <div class="form-group" th:classappend="${#fields.hasErrors('id_pedido')} ? has-error">
                        <label for="id_pedido" class="col-sm-2 control-label">Código Interno</label>
```

```
                    <div class="col-sm-2">
                    <input type="text" disabled="disabled" class="form-control" id="id_pedido"
th:field="*{id_pedido}"/>
                    </div>
                    </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('numero')} ? has-error">
                    <label for="numero" class="col-sm-2 control-label">Número</label>
                    <div class="col-sm-2">
                              <input type="text" class="form-control" id="numero" th:field="*{numero}"/>
                    </div>
                    </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('segurado')} ? has-error">
                    <label for="segurado" class="col-sm-2 control-label">Nome Segurado</label>
                              <div class="col-sm-4">
                              <input type="text" class="form-control" id="segurado" th:field="*{segurado}"/>
                              </div>
                    </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('dataCadastro')} ? has-error">
                              <label for="dataCadastro" class="col-sm-2 control-label">Dt. Cadastro</label>
                    <div class="col-sm-2">
                    <input type="date" class="form-control" id="dataCadastro" th:field="*{dataCadastro}"/>
                    </div>
                    </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('dataPrazo')} ? has-error">
                              <label for="dataPrazo" class="col-sm-2 control-label">Dt. Prazo</label>
                    <div class="col-sm-2">
                    <input type="date" class="form-control" id="dataPrazo" th:field="*{dataPrazo}"/>
                    </div>
                    </div>
                    <div class="form-group">
                              <label for="seg" class="col-sm-2 control-label">Inspetor</label>
                              <div class="col-sm-2">
                              <select class="form-control" name="inspetor" th:field="*{inspetor.id_pessoa}">
                              <option th:each="inspetor : ${todasPessoas}" th:value="${inspetor.id_pessoa}"
                                                                                  th:text="${inspetor.nome}">
          </option>
                    </select>
          </div>
          </div>
                    <div class="form-group">
                              <label for="status" class="col-sm-2 control-label">Seguradora</label>
                              <div class="col-sm-2">
                                        <select class="form-control" name="cia" th:field="*{cia.id_pessoa}">
                                        <option th:each="cia : ${todasPessoas}" th:value="${cia.id_pessoa}"
th:text="${cia.nome}">
          </option>
                              </select>
          </div>
                              </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('foneSegurado')} ? has-error">
                              <label for="foneSegurado" class="col-sm-2 control-label">Fone Segurado</label>
                              <div class="col-sm-2">
                              <input type="text" class="form-control" id="foneSegurado"
th:field="*{foneSegurado}"/>
                              </div>
                    </div>
                    <div class="form-group" th:classappend="${#fields.hasErrors('endSegurado')} ? has-error">
                              <label for="endSegurado" class="col-sm-2 control-label">End. Segurado</label>
                              <div class="col-sm-4">
```

```
                                        <input type="text" class="form-control" id="endSegurado"
th:field="*{endSegurado}"/>
                                </div>
                        </div>

                <div class="form-group">
                <label for="anexoPedido" class="col-sm-2 control-label">Anexos</label>
                <div class="col-sm-2">
                <input type="file" id="anexos" name="anexos" multiple="multiple" />
                </div>
                <br />
                <ul>
                        <li th:each="anexo : ${anexos}">
                        <a
th:href="@{../../anexos/{id_pedido}/{nome}(id_pedido=${pedido.id_pedido},nome=${anexo.nome})}"th:text=
"${anexo.nome}"></a>
                        </li>
                                </ul>
                </div>
…
```

Listagem 7 – Exemplo para criar campo na tela de pedido

Na listagem 7 os campos “inspetor” e “seguradora” são carregados pela *tag* “<select>”, que busca o nome da pessoa diretamente da tabela pessoa. Ao editar um cadastro, como apresentado na Figura 6, os campos são carregados e preenchidos automaticamente, pois os dados do pedido a ser editado são previamente carregados na classe *controller*.

Conforme descrito no subcapítulo 2.1.2 o Maven facilita o gerenciamento de dependências, evitando que o desenvolvedor importe arquivos de dependência de forma manual. Na Listagem 8 está parte do código que faz parte do arquivo pom.xml utilizado pelo Maven para a importação e gerenciamento das dependências. Essas dependências são referentes ao *driver* de conexão com banco de dados utilizado, Thymeleaf, segurança, entre outros.

```
…
<parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.5.4.RELEASE</version>
                <relativePath/><!-- lookup parent from repository -->
        </parent>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
```

```
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-thymeleaf</artifactId>
                </dependency>
...
```

Listagem 8 – Dependências Maven

4 CONSIDERAÇÕES FINAIS

Apesar de estarmos vivendo em uma era tecnológica pode-sefrequentemente se depararcom funções que pouco exploram os recursos tecnológicos disponíveis. Neste trabalho identificou-se que o segmento de vistorias de seguros é um deles e que soluções tecnológicas podem trazer mais comodidade, agilidade e segurança. Foi desenvolvido nesse trabalho um software web que faz o cadastro do pedido de vistoria com principais dados e também permite anexar os arquivos envolvidos em cada vistoria, centralizando assim em um local todas as informações, que tanto inspetores quanto administradores tenham acesso.

Para o sistema desenvolvido neste trabalho, pode-se observar que o Spring Framework atende muito bem a necessidade, com uma boa produtividade diminui o tempo de desenvolvimento, e ao mesmo tempo traz segurança a aplicação. Também o Bootstrap e Thymeleaf permitem definir um leiaute com boa usabilidade.

As principais dificuldades encontradas durante o desenvolvimento foram realizar todas as configurações iniciais do projeto, porém, o próprio site do Spring permite gerar o projeto com todas as configurações iniciais necessárias, basta importar para o ambiente de desenvolvimento.

Como trabalhos futuros visando complementar parte do que já foi implementado podem ser criadas novas funcionalidades para controle de comissões de inspetores, gerenciamento de áreas de atuação e criação de questionários que possam ser integrados diretamente com o sistema das seguradoras.

REFERÊNCIAS

AFONSO, Alexandre.**Desenvolvimento Web com Spring**. Uberlândia: AlgaWorks Softwares, Treinamentos e Serviços Ltda., 2017.

FENASEG, 2017

MAVEN, **Apache Maven Project,** 2017. Disponível em: <https://maven.apache.org/what-is-maven.html>. Acesso em: 12 dez. 2017.

PERIM, D.M. Seguros: uma visão histórica e conceitual. Monografia (Graduação em Administração de Empresas). Vitória, ES: Universidade Federal do Espírito Santo, 2002.

POSTGRESQL, **Sobre o PostgreSql,** 2017. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 01 dez. 2017.

RUFFAT, M.; CALONI, E.V; LAGUERRE, B. L'UAP et l'histoire de l'assurance – Maison dês Sciences de L'Homme – Editions Jean-Claude Lattés, 1990.

SUSEP, **6º Relatório de análise e acompanhamento dos mercados supervisionados,** 2016. Disponível em: <http://www.susep.gov.br/menuestatistica/SES/Relat_Acomp_Mercado_2016.pdf>. Acesso em: 01 dez. 2017.

TIMPE, **Thymeleaf,** 2017. Disponível em: <http://www.thymeleaf.org/>. Acesso em: 01 dez. 2017.