UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
III ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

ADRIANE DE COL

SISTEMAS MULTI-TENANCY: DESENVOLVIMENTO COM SPRING, HTML 5 E BOOTSTRAP

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

ADRIANE DE COL

SISTEMAS MULTI-TENANCY:
DESENVOLVIMENTO COM SPRING, HTML 5 E BOOTSTRAP

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, campus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Profa. Andreia Scariot Beulke.

PATO BRANCO
2015

SISTEMAS MULTI-TENANCY: DESENVOLVIMENTO COM SPRING, HTML 5 E BOOTSTRAP

Por

Adriane de Col

Esta monografia foi apresentada às 17h30 do dia 09 de setembro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. A acadêmica foi arguido pela Banca Examinadora composto pelo professor abaixo assinado. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profa. Esp. Andreia Scariot Beulke
Orientadora
UTFPR – Campus Pato Branco

Prof. Msc. Vinicius Pegorini
Banca
UTFPR – Campus Pato Branco

Profa. Dr. Beatriz Terezinha Borsoi
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Roosevelt Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

RESUMO

COL, Adriane De. Sistemas *multi-tenancy*: desenvolvimento com Spring, HTML 5 e Bootstrap. 2015. 73 f. Monografia de especialização. III Curso de Especialização em Java. Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

As aplicações web responsivas fazem parte de uma evolução tecnológica que permite a disponibilização de interfaces adaptáveis independente da resolução do ambiente utilizado para acesso. O *framework* Bootstrap combinado com a linguagem de marcação HTML 5 possibilita o desenvolvimento desse tipo de *user interface*. Por sua vez, a utilização da arquitetura *multi-tenancy* combinada com o padrão de projetos MVC (*Model-View-Controller*) permite a criação de um projeto padronizado e bem estruturado. Contudo, é necessário que o desenvolvimento seja realizado de maneira simples e ágil e nesse aspecto é possível contar com as tecnologias disponíveis nos *frameworks* Spring e Apache Maven. O Apache Maven atua desde o controle de dependências até o *deploy* da aplicação, enquanto o Spring disponibiliza praticidade na realização de consultas e conexão com o banco de dados e o controle de autorização e autenticação. Visando exemplificar o uso dessas tecnologias, este trabalho apresenta o desenvolvimento de um sistema web responsivo para realizar o controle de entregas em domicílio.

Palavras-chave: Bootstrap. Spring. Maven. Java. Multi-tenancy.

ABSTRACT

COL, Adriane De. Multi-tenancy systems: development with Spring, HTML 5 and Bootstrap. 2015. 73 f. Monografia de especialização. III Curso de Especialização em Java. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2015.

The responsive web applications are part of a technology evolution that enables the provision of adaptive interfaces regardless of the resolution of the environment used to access. The Bootstrap framework combined with the markup language HTML 5 enable the development of such user interface. In turn, the use of multi-tenancy architecture combined with the MVC project (Model-View-Controller) allows the creation of a standardized and structured design. However, it is necessary that development be done in a simple and flexible manner and in this respect we can rely on the technologies available in the frameworks Spring and Apache Maven. The Apache Maven operates from the premises of control to deploy the application, while Spring provides convenience in consultations and connecting to the database and authorization control and authentication. Aiming to exemplify the use of these technologies, this paper presents the development of a responsive web system to carry out the home delivery.

Palavras-chave: Bootstrap. Spring. Maven. Java. Multi-tenancy.

LISTA DE FIGURAS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

AOP	*Aspect-Oriented Programming*
API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
DER	Diagrama de Entidades e Relacionamentos
ERP	*Enterprise Resource Planning*
FTP	*File Transfer Protocol*
GUI	*Graphical User Interface*
GWT	*Google Web Toolkit*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JAVA EE	*Java Enterprise Edition*
JAVA SE	*Java Standard Edition*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JMS	*Java Message Service*
JPA	*Java Persistence API*
MVC	*Model View Controller*
ORM	*Object Relational Mapping*
SAAS	*Software as a Service*
SEO	*Search Engine Optimization*
SGBD	Sistema de Gerenciamento de Banco de Dados
TCP	*Transmission Control Protocol*
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
XHTML	*Extensible HyperText Markup Language*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa deste trabalho.

1.1 CONSIDERAÇÕES INICIAIS

Em qualquer área de negócio a inovação é um diferencial, pois proporciona às empresas mais dinamismo e vantagens competitivas. O conceito de inovação apresenta variações e depende do contexto de sua aplicação. Na área tecnológica, por exemplo, a inovação se aplica na implantação e melhoria de produtos e processos. De acordo com Laudon e Laudon (2007), a Internet e suas tecnologias emergentes se destacam pela inovação de produtos e processos por serem fontes de melhoria da competitividade no mercado globalizado. Existe uma variedade de sistemas e aplicativos disponíveis na Internet. Um exemplo desses sistemas são os que oferecem serviços de pedidos, agendamento e gerenciamento de entrega de produtos *online*. Muitas empresas, como, por exemplo, restaurantes, bares, entre outros, estão adquirindo aplicativos web para otimizar os pedidos e entrega de seus produtos. Empresas que realizam entregas com agendamento somente via atendimento telefônico deixam de atingir usuários que estão habituados à utilização da Internet para este fim, e assim, perdem em produtividade e qualidade no atendimento a seus clientes. Portanto, as empresas podem inovar ao utilizarem sistemas e aplicativos *online* para realizar pedidos, agendar e gerenciar a entrega de produtos, proporcionando, assim, agilidade e qualidade nos serviços prestados.

Utilizando um aplicativo de controle de entrega é possível otimizar os processos do estabelecimento. Dessa forma, o cliente pode realizar e verificar o andamento dos seus pedidos, bem como o usuário retaguarda pode acompanhar todos os pedidos e atualizá-los.

Nesse viés, pode-se afirmar que devido à grande variedade de dispositivos utilizados para acesso à Internet, como, por exemplo, *smartphones*, *tablets* e celulares, é importante que os sistemas sejam desenvolvidos com tecnologias e

processos adequados, para que possam, efetivamente, atender as necessidades dos seus usuários.

Como a interface é o canal de comunicação entre o usuário e o sistema, é importante que ela seja desenvolvida visando facilitar a interação para que o usuário mantenha a atenção em requisitos inerentes às tarefas que deseja realizar. Assim, observa-se que, com a crescente demanda da web *mobile*, tornou-se imprescindível a adaptação dos sistemas aos diversos tamanhos de telas devido ao uso de diferentes dispositivos para acesso à Internet.

Com o intuito de agregar inovação e otimizar processos aos estabelecimentos de entrega em domicílio, este trabalho se propõe a implementar um sistema web de controle de entregas, que será utilizado por usuários que possuem dispositivos diversos. Portanto, para implementar telas responsivas, ou seja, que se adaptam a diferentes dispositivos, será utilizado o *framework front-end* Boostrap, a linguagem de marcação *Hipertext Markup Language* (HTML), a linguagem de formatação *Cascading Style Sheet* (CSS) e a linguagem de *script* Javascript. Para agilizar e simplificar o processo de desenvolvimento *back-end* será utilizado o *framework* Spring.

1.2 OBJETIVOS

A seguir será apresentado o objetivo geral, bem como os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

Implementar um sistema web de controle de entregas utilizando as tecnologias HTML 5, Spring e Bootstrap 3.

1.2.2 Objetivos Específicos

- Apresentar a utilização dos *frameworks* Apache Maven e Spring no desenvolvimento *back-end*;

- Apresentar o uso da arquitetura *multi-tenancy* no desenvolvimento de um sistema web;
- Desenvolver um sistema responsivo com HTML 5 e Bootstrap na implementação do *front-end.*

1.3 JUSTIFICATIVA

Um aplicativo que permita ao usuário final realizar pedidos pela web, seja utilizando dispositivos como o *notebook, smartphone* ou qualquer outro dispositivo móvel com acesso a um navegador e Internet, visa facilitar e otimizar os processos de pedido e entrega do produto. Tem sido cada vez mais comum muitas pessoas, motivadas pela falta de tempo, encontrarem na entrega em domicílio a melhor forma de se alimentar sem ter que enfrentar filas, falta de estacionamento, entre outros. A facilidade de acesso à web, em especial à crescente web *mobile*, tornou possível a realização dessa tarefa rapidamente e com poucas interações.

Na área empresarial, fornecer um sistema web flexível, com tecnologias atuais e de ótima qualidade é considerado estratégico. Dentre os aspectos positivos, além de estreitar a relação entre cliente e fornecedor, este diferencial fideliza o cliente, tendo em vista que muitos sistemas disponíveis são de difícil utilização, forçando os usuários a procurar alternativas de contato com a loja.

Diante da crescente quantidade de dados que as operações comerciais e de negócio geral, muitas das quais precisam ser gerenciadas pelas empresas, poder contar com um sistema que otimiza os controles internos e auxilia seus gerentes na tomada de decisões fornecendo informações relevantes aumenta a competitividade e destaca as empresas no mercado.

No desenvolvimento deste trabalho, optou-se pela utilização dos *frameworks* Apache Maven e Spring para a construção do *back-end* em decorrência da facilidade e agilidade promovidas ao desenvolvimento. Em contrapartida, a adoção da linguagem de formatação HTML 5 e o *framework front-end* Boostrap permite a criação de interfaces simples e agradáveis e que conta ainda com os recursos da responsividade, proporcionando mais dinamismo e conforto no acesso das informações das páginas web. Dessa forma, este trabalho exemplifica o

desenvolvimento de um sistema responsivo e desenvolvido utilizando a linguagem Java com os *frameworks* Spring, Apache Maven e HTML 5 com Boostrap.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho com os seguintes temas: sistemas web, arquiteturas 2, 3 e 4 camadas, *multi-tenancy*, responsividade e HTML 5. Esses tópicos são abordados por que o resultado deste trabalho é um sistema web implementado com recursos de interface responsiva e HTML 5, o qual necessita ser desenvolvido com uma estrutura de múltiplas camadas visando atender a vários clientes simultâneos.

2.1 SISTEMAS WEB

De acordo com Paula Filho (2003), aplicações web são produtos de *software* que utilizam uma arquitetura distribuída sob protocolo *HyperText Transfer Protocol* (HTTP). Por consequência, a interface com o usuário é acessada a partir de um cliente, normalmente um navegador (*browser*), o qual realiza solicitação a determinado servidor. Essas requisições são efetivadas através da invocação de *Uniform Resource Locator (*URLs), que representam a localização do recurso.

O protocolo HTTP descreve as regras de comunicação entre o cliente e servidor, sendo o responsável por realizar o pedido do cliente e retornar a resposta do servidor, processo denominado de *request*/*response* (OSORIO, 2013). De acordo com Kurose (2006) este protocolo estabelece a estrutura e o modo de tratamento de mensagens trocadas e é dividido em duas partes (cliente e servidor) que são executadas em computadores distintos.

Os navegadores são responsáveis por renderizar os conteúdos da web, ou seja, exibi-los para o usuário. As especificações de como esse processo é realizado são mantidas pelo *World Wide Consortium* (W3C), organização responsável pela manutenção dos padrões da web. O W3C tem como missão levar a *World Wide Web* (WWW) ao seu potencial máximo, desenvolvendo protocolos e diretrizes que garantam seu crescimento a longo prazo (W3C, 2011). Por muito tempo houve imparcialidade por parte dos navegadores no cumprimento dos padrões desenvolvidos pela W3C, fato que gerava problemas de incompatibilidade nos

sistemas (IRISH, GARSIEL, 2011). A Figura 11 mostra o mecanismo de renderização do conteúdo web.

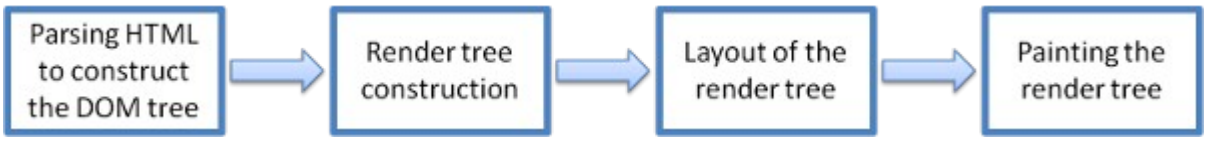

Figura 1 - Fluxo básico do mecanismo de renderização
Fonte: Irish e Garsiel (2011).

Diante da necessidade de melhoria na estilização e renderização dos conteúdos web, visto que as páginas desenvolvidas com a linguagem HTML apresentavam um layout simplório e pouco elegante, surgiu o CSS, que são folhas de estilo em cascata que permitem adicionar estilos aos documentos web (W3C, 2011). Inicialmente implementado apenas para melhorar o visual das páginas, como troca de cores, formato de fontes, etc, o CSS permite que as marcações utilizadas em uma página possam ser apresentadas em diferentes estilos, conforme os métodos de renderização (como em uma tela, impressão, via voz, baseadas em dispositivos táteis, etc.) (PEREIRA, 2009).

Em 2008 houve o lançamento da quinta versão do HTML, que é a versão atual da linguagem. Simoni (2013) afirma que a criação do HTML 5 visa agregar significado semântico às páginas web e padronizar o entendimento das seções de um site pelos navegadores.

O HTML 5 tem sido bastante utilizado pelos desenvolvedores, uma vez que além de fornecer melhor estruturação e entendimento da página, possui carregamento mais rápido em comparação às versões anteriores e é uma tendência do mercado.

A concepção e desenvolvimento de um sistema web, independente das tecnologias adotadas, precisa ter estruturas bem definidas tanto no lado cliente quanto no lado servidor, assunto explanado na seção 2.2.

2.2 ARQUITETURAS EM CAMADAS

Segundo Costa (2014), arquitetura é a construção concebida visando ordenar e organizar determinada estrutura com finalidade e intensão específicas.

Existem dois componentes essenciais ao se falar em arquitetura: o servidor – equipamento com maior poder de processamento e armazenamento – e o cliente, microcomputador que se conecta ao servidor vida rede. O cliente tem como papel iniciar e finalizar as transações, portanto pode ser um dispositivo com hardware simples. Já o servidor recebe e responde solicitações, prestando serviços distribuídos e atendendo múltiplos clientes simultaneamente, fato que exige uma máquina mais robusta.

Na arquitetura 2 camadas, também conhecida por arquitetura cliente-servidor, as regras de negócio ficam no lado cliente, bem como a responsabilidade de apresentar a interface para o usuário, como formulários, menus, entre outros. Nesse modelo um aplicativo é instalado em cada máquina cliente e faz acesso direto ao banco de dados. Com o passar do tempo e a evolução do mercado, essa arquitetura mostrou-se de difícil manutenção, surgindo a proposta de 3 camadas. A Figura 2 mostra o modelo cliente-servidor.

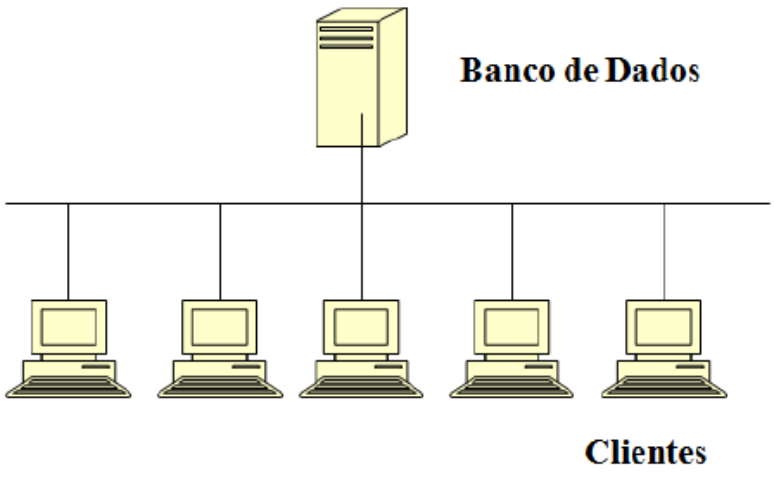

Figura 2 - Modelo cliente servidor

Na tentativa de resolver os problemas do modelo 2 camadas é que foi criado o conceito de servidor de aplicação. A ideia dessa 3ª camada é centralizar a lógica e regras de negócio, bem como o acesso ao banco de dados. A responsabilidade pela

apresentação do sistema continua nos clientes, o que mantém a exigência de atualização de todas as máquinas (BATTISTI, 2003). A Figura 3 exibe o modelo de arquitetura 3 camadas.

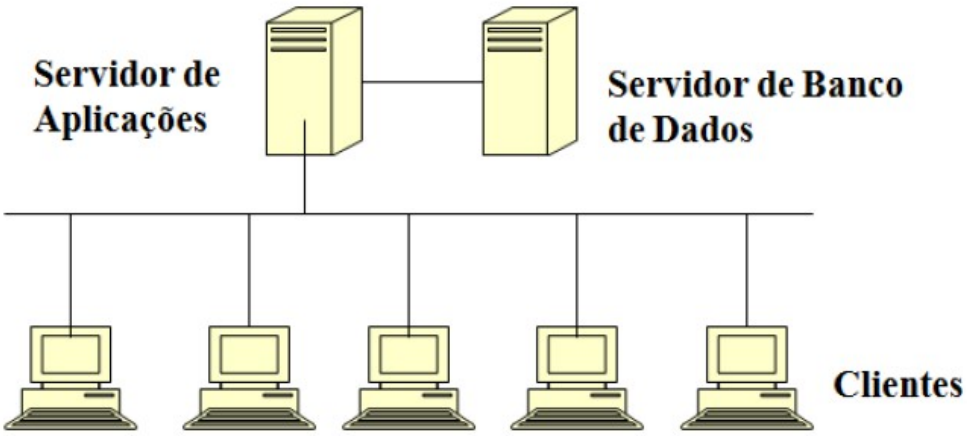

Figura 3 - Arquitetura 3 camadas

A evolução para o modelo 3 camadas resolveu alguns problemas da proposta cliente-servidor, porém manteve a necessidade de atualização das máquinas clientes. Ante esse desafio é que surgiu o modelo 4 camadas, o qual propõe a retirada da apresentação do cliente e a centralização em um único ponto, normalmente um servidor web. Com essa estrutura de construção o cliente deixa de existir como um programa a ser instalado em cada computador e passa a ser o navegador, como o Google Chrome por exemplo. Esse formato tornou o processo de atualização dos aplicativos gerenciável, uma vez que não é mais necessário fazer isso em centenas ou milhares de computadores. O acesso à aplicação em uma arquitetura 4 camadas é feita através de um endereço, denominado URL. A Figura 4 mostra o conceito da arquitetura de 4 camadas.

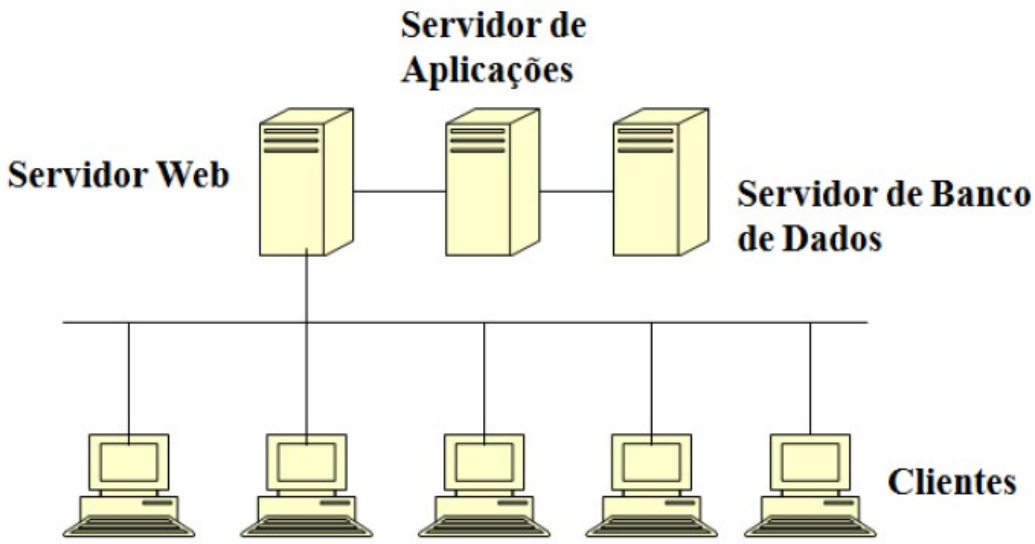

Figura 4 - Arquitetura 4 camadas

2.3 MULTI-TENANCY

Há alguns anos surgiu o conceito de computação em nuvem, o qual se refere à utilização de computadores compartilhados e conectados via Internet que permitem que uma aplicação possa ser acessada de maneira remota de qualquer lugar através da Internet. Uma das formas de distribuição e comercialização de *software* baseada nesse conceito é o *Software as a Service* (SaaS), onde o fornecedor é o responsável por toda a estrutura necessária para a disponibilização do sistema, deixando a cargo do cliente somente a utilização do mesmo via web. A principal proposta desse modelo é ter somente uma aplicação atendendo vários clientes, chamados de inquilinos ou *tenants*. A Figura 5 exibe a arquitetura *multi-tenancy*.

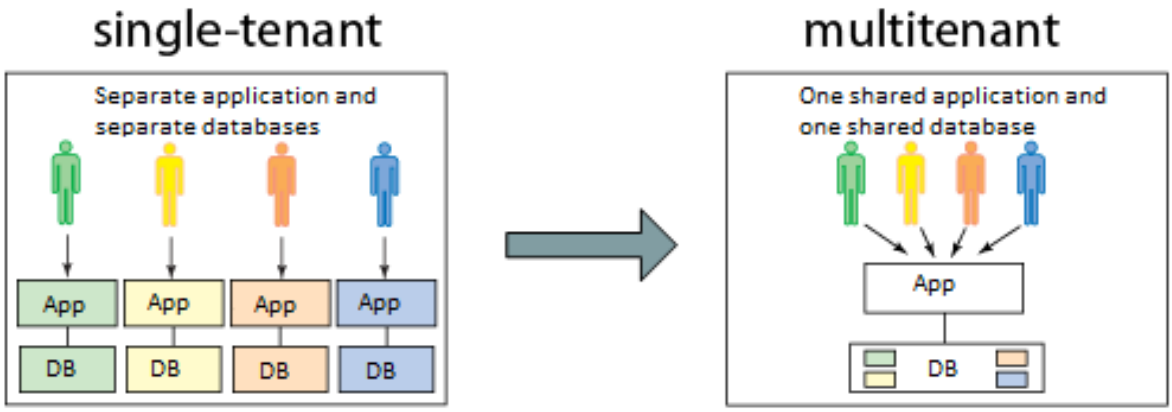

Figura 5 - Arquitetura *Multi-Tenancy*
Fonte: Adhikari (2015).

A arquitetura *multi-tenancy* é uma arquitetura essencial para um ambiente em nuvem, pois permite que múltiplos inquilinos compartilhem os mesmos recursos físicos como um aplicativo ERP (*Enterprise Resource Planning*), mas permaneçam logicamente isolados (TAURION, 2010).

Conforme afirma Silveira (2010), podem existir vários tipos de abordagens do *multi-tenancy* conforme o nível de compartilhamento de recursos. Por um lado é possível manter todos os clientes isolados, bem como compartilhar todos os recursos. No segundo caso é preciso que haja garantias de segurança das informações de forma que nenhum acesso não autorizado ocorra. Outro aspecto relevante dessa abordagem ocorre no âmbito das customizações, visto que todos os clientes compartilham o mesmo servidor web, dificultando tal implantação.

2.4 RESPONSIVIDADE

De acordo com Altermann (2012), não existe novidade no conceito de responsividade, porém o assunto tem chamado bastante atenção nos últimos tempos. Um dos motivos é o crescimento do mercado *mobile*, que proporciona telas com resoluções e tamanhos variados.

O *design* responsivo é uma evolução tecnológica de *design* de sites e foi criado em maio de 2010 por Ethan Marcotte com o objetivo de oferecer um site único para todos os tipos de acessos contando com o auxílio de técnicas de CSS, ou seja,

fornecer adaptação a todas as resoluções e com isso melhor experiência ao usuário (LOPES, 2012). A Figura 6 exibe o comportamento do *design* responsivo.

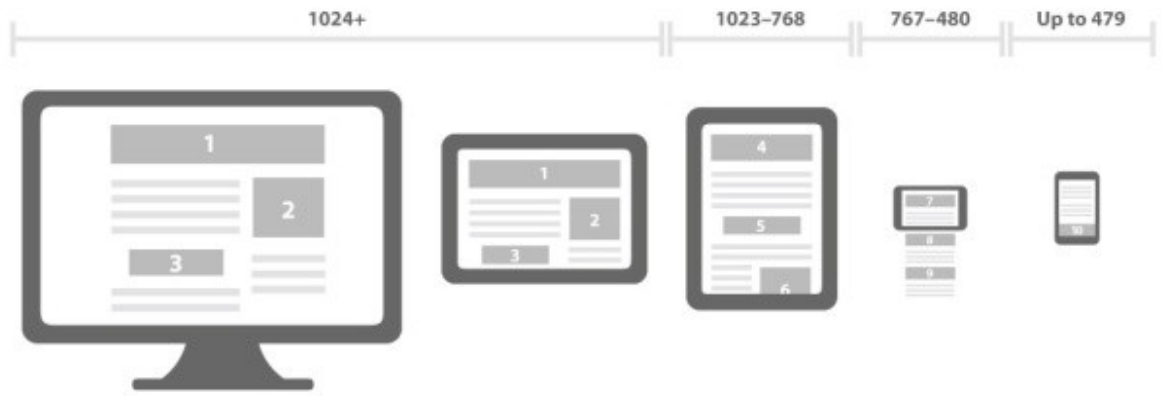

Figura 6 - Comportamento do layout responsivo
Fonte: Raddar (2015).

Silva (2015) aponta que o termo responsivo vai além de permitir que os *layouts* se adaptem a diferentes resoluções, pois deve responder às características dos diversos dispositivos em que o aplicativo web é acessado, ou seja, o *layout* responsivo deve contrair-se e expandir-se com a finalidade de disponibilizar o conteúdo web de maneira que seja acessível na área onde é visualizado.

Conforme afirma Lopes (2015), as principais ferramentas de desenvolvimento de um *layout* responsivo são:

1 – *Media Queries*: disponíveis no CSS 3, possibilitam a criação de regras que serão aplicadas somente em situações determinadas. Podem ser definidas para resoluções de telas específicas, conforme a orientação do dispositivo ou a densidade de *pixels*.

2 – *Layout* Fluído: caracteriza o desenvolvimento sem a utilização de *pixels*, mas sim um layout flexível baseado em percentuais e EMs[1].

3 – Imagens Flexíveis: para conquistar um layout fluído é necessário que as imagens se adaptem às resoluções. Nesse quesito é necessário um cuidado maior, pois como as imagens são baseadas em *pixels*, usar percentuais para redimensioná-las pode causar distorções, como um layout pixelizado ao aumentá-la.

1Unidade de medida relativa, que varia proporcionalmente conforme o contexto.

Por outro lado, carregar uma imagem de alta resolução para depois diminuí-la deixará a página lenta. Ou seja, é necessário trabalhar com as imagens em várias resoluções. A utilização de *media queries* pode auxiliar nesse aspecto.

A web design responsiva conquistou definitivamente seu espaço quando a Google passou a recomendá-la (SEO, 2014), bem como afirmou com o termo "*mobile-friendly*" que a sua utilização é considerada para posicionamento de sites.

2.5 HTML 5

O HTML nasceu em 1992 e desde então está em constante evolução. Em 1998 a versão 4 do HTML tornou-se uma recomendação da W3C, e visava atender as necessidades da web estática (TERUEL, 2011). Assim, era utilizada para desenvolver aplicativos web com conteúdos estáticos, sem que o usuário pudesse interagir dinamicamente.

Ao perceber que a versão 4 não atendia as necessidades de desenvolvimento, a W3C criou a versão *Extensible Hipertext Markup Language* (XHTML) que trabalha com regras da linguagem *Extensible Markup Language* (XML). No entanto, essa versão também não atendeu aos anseios da comunidade desenvolvedora de páginas web. E, para tanto, algumas empresas e desenvolvedores criaram o *Web Hypertext Application Technology Working Group* (WHATWG), "uma organização com foco no trabalho com formulário web e aplicações" (TERUEL, 2011, p. 19). Algum tempo depois a W3C fez parceria com o WHATWG e, assim, passaram a desenvolver juntos o HTML 5.

A versão 5 do HTML foi lançada em 2008, e seu principal foco é fornecer significado semântico às páginas web. Teruel (2011) afirma que a versão 5 do HTML é "a maior revisão da linguagem HTML que será o novo padrão para web que une o que há de melhor no HTML versão 4, XHTML e HTML DOM e adiciona um conjunto de novo recursos". A versão 5 do HTML conta com uma estrutura mais objetiva e de fácil compreensão, cada seção de uma página web tem um significado de acordo com a sua *tag*. A Figura 7 mostra a estrutura de uma página em linguagem HTML 5.

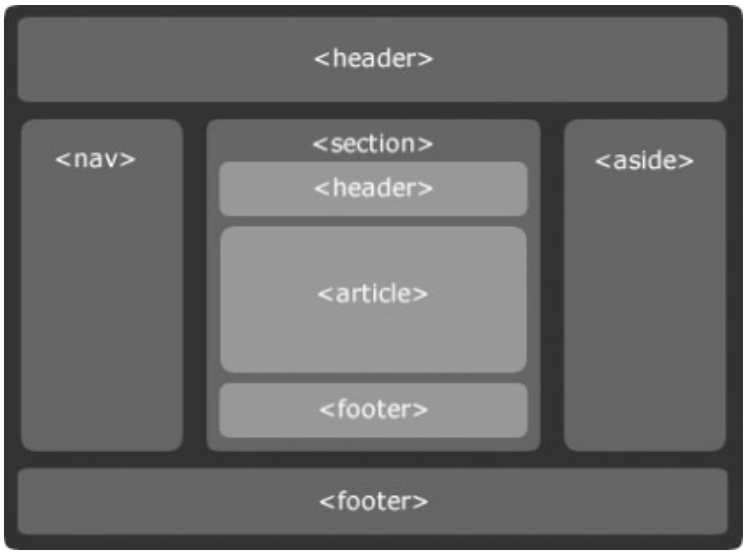

Figura 7 - Estrutura básica do HTML 5
Fonte: Simoni (2013).

A versão 5 do HTML disponibiliza vários novos elementos e recursos para divisão dos conteúdos web, criação de *layouts* e formulários, como por exemplo, campos que permitem a captura de data e hora do sistema ou de um calendário, campos para entrada de dados de *email*, telefone, números, entre outros. A HTML 5 permite, também, a utilização de uma série de APIs que auxiliam na criação de aplicações web e podem ser acessadas utilizado JavaScript e utilizadas em conjunto com novos elementos. De acordo com Teruel (2011) essas APIs são:

a) O Webforms 2.0 que disponibiliza mais tipos de entrada de dados em formulários, validação de entrada mais simples e menos *scripts* de página.

b) Web SQL Database que fornece um conjunto de APIs para manipular banco de dados do lado cliente usando SQL via JavaScript.

c) Web Storage que permite manipular o histórico de navegação e dados no computador cliente utilizando uma forma mais eficiente do que os antigos *cookies*.

d) Web Sockets que permite que páginas web usem o protocolo Web Socket para a comunicação de duas vias com um servidor.

e) Web Works utilizada para a execução de *scripts* em segundo plano, independentemente de qualquer *script* em execução na interface de usuário.

f) Geolocation que define uma interface de alto nível para as informações de localização associada apenas com o dispositivo que hospeda a aplicação, como latitude e longitude.

g) Vídeo e áudio para executar conteúdos multimídia.

h) Canvas e SVG para permitir desenhar na tela do navegador.

i) Aplicações off-line para executar aplicações web referenciadas em cache pelo atributo *manifest* do elemento <html>.

j) *Drag and drop* que permite operações para arrastar e soltar elementos na tela do navegador.

Teruel (2011) menciona que dentre as principais vantagens e novidades do HTML 5 estão: redução da necessidade de *plugins* externos, disponibilização de elementos semelhantes em diversos tipos de navegadores e dispositivos e não permite que elementos de estilização e formatação sejam utilizados com a marcação. Portanto, o autor recomenda que a estilização e formatação seja desenvolvida por meio do CSS.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os materiais e métodos utilizados no desenvolvimento deste trabalho.

3.1 MATERIAIS

As ferramentas e *frameworks* utilizados para a implementação das atividades de modelagem, implementação e execução do sistema são:

a) MySQL Workbench 6.2 para modelagem de banco de dados;

b) Java para linguagem de desenvolvimento;

c) IntelliJ IDEA 14.1.2 como *Integrated Development Environment* (IDE) e para modelagem de classes;

d) Spring Framework, Spring Data JPA e Spring Security;

e) MySQL 5.6 como SGBD (Sistema de Gerenciamento de Banco de Dados);

f) Apache Maven para gerenciamento de projeto;

g) WildFly 8.1 como servidor de aplicação web;

h) Hibernate para mapeamento objeto relacional;

I) HTML 5 e Boostrap para GUI (*Graphical User Interface*).

3.1.1 MySQL Workbench

MySQL Workbench é uma ferramenta que possibilita o gerenciamento visual de base de dados MySQL. Por meio dessa ferramenta é possível criar a modelagem de dados, diagramas de entidade relacionamento, realizar engenharia reversa e a migração de dados de outras bases, como Microsoft SQL Server. (WORKBENCH, 2015). A Figura 8 exibe a tela do MySQL Workbench.

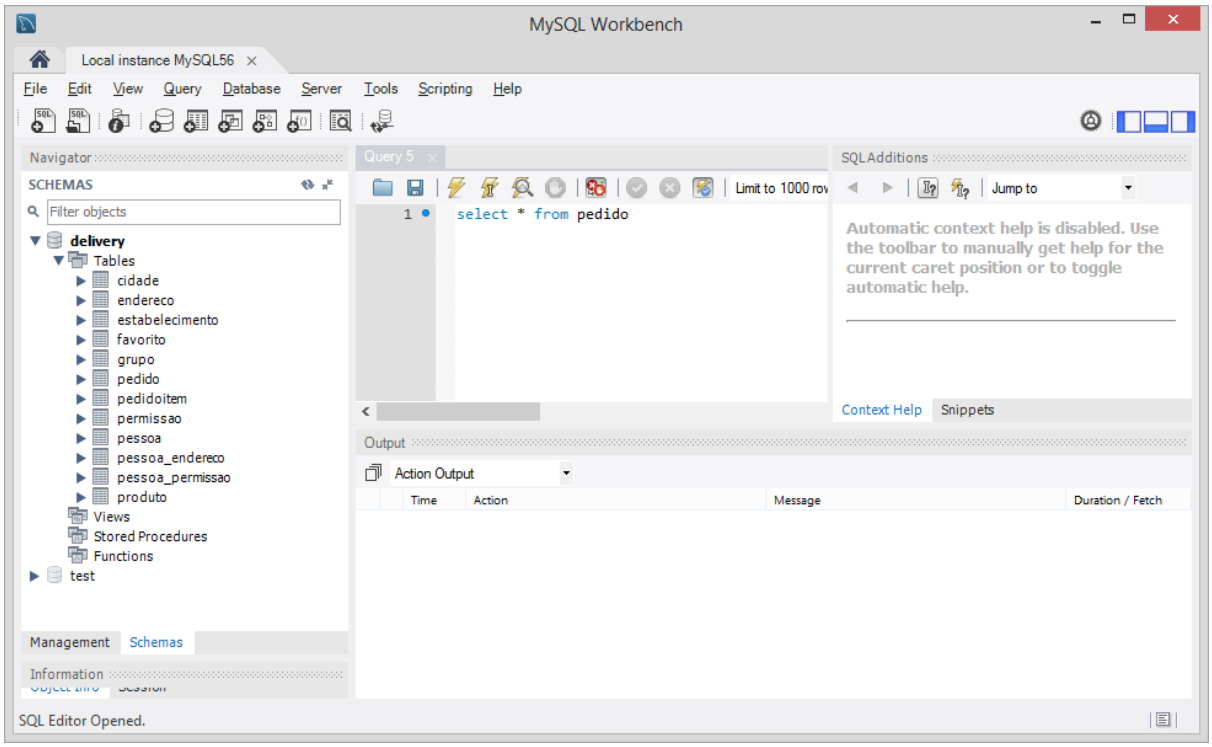

Figura 8 - MySQL Workbench

3.1.2 Linguagem Java

A linguagem Java foi criada na década de 1990 por uma equipe liderada por James Gosling na empresa Sun Microsystems, a qual foi adquirida pela Oracle em 2009. A resolução de alguns problemas de desenvolvimento da época motivaram a sua origem, dentre eles estão os ponteiros, o gerenciamento de memória e a necessidade de readaptação dos sistemas por mudanças de sistemas operacionais (JAVA, 2015).

As principais características da linguagem Java são:

1 – Orientação a Objetos: à exceção dos tipos primitivos, como *int* e *float*;

2 – Portabilidade: independência de plataforma operacional;

3 – Recursos de rede: biblioteca de grande porte para protocolos TCP/IP (*Transmission Control Protocol*/*Internet Protocol*) como o HTTP e FTP (*File Transfer Protocol*);

4 – Segurança: permite a execução de sistemas via rede com restrições.

3.1.3 IntelliJ IDEA

O IntelliJ IDEA é um *Integrated Development Environment* (IDE) para a linguagem Java, Scala e Groovy mantido pela JetBrains. Oferece suporte a diversos *frameworks*, como por exemplo Spring, Google Web Toolkit (GWT), Apache Maven, Gradle, dentre outros. Além dos sistemas em diversas linguagens possibilita o desenvolvimento de aplicativos móveis em Android, PhoneGap, Cordova e Ionic. No escopo web provê suporte para JavaScript, HTML e CSS, incluindo AngularJs e Node.js (INTELLIJ, 2015).

A adoção do IntelliJ como IDE tem crescido bastante nos últimos tempos especialmente nas corporações, por se tratar de uma ferramenta moderna e com suporte nativo a muitos *frameworks*. Em IntelliJ é possível realizar o *download* da versão gratuita do *software*, o qual possui algumas limitações, bem como da versão paga com um *trial*[2] de 30 dias.

O produto possui algumas opções de licenciamento diferenciadas, como a exemplo dos valores especiais para desenvolvedores individuais e *startups* e versões gratuitas para estudantes e professores, situações de uso educacional e projetos *open source*. A Figura 9 exibe a tela da IDE.

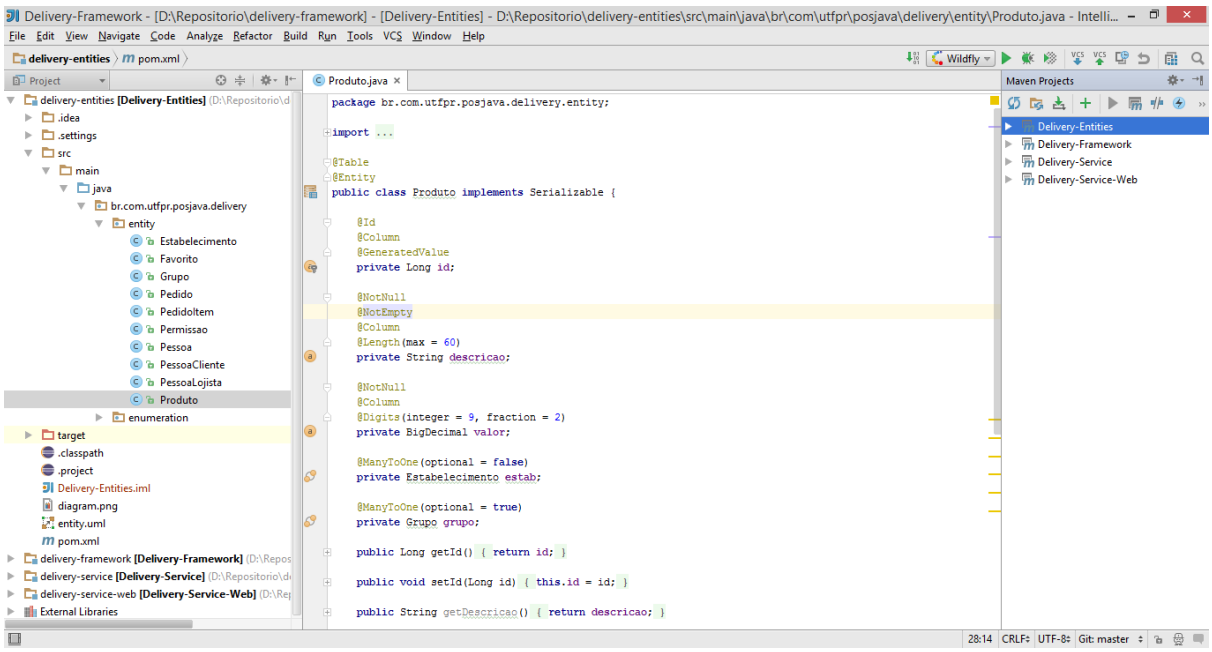

Figura 9 - IntelliJ IDEA

2 Programa com funcionalidades disponíveis por um período determinado.

3.1.4 Spring Framework

O Spring é um *framework* de código aberto criado por Rod Johnson visando simplificar o desenvolvimento em Java. Conta com diversos módulos, como o Spring Data e Security, sendo que alguns deles são aplicáveis somente ao desenvolvimento *web*, porém o principal pode ser aplicado em qualquer aplicativo Java (GENTIL, 2015).

A biblioteca Spring Framework fornece um modelo de desenvolvimento e configuração de aplicativos modernos baseados em Java. O foco do Spring é montar a estrutura para os aplicativos de maneira que as equipes fiquem centradas nas regras de negócio do sistema (SPRING FRAMEWORK, 2015).

O Spring Framework fornece injeção de dependência, *Aspect-Oriented Programming* (AOP), desenvolvimento facilitado do padrão *Model View Controller* (MVC) para aplicações Web e *RESTful*, suporte à *Java Database Connectivity* (JDBC), *Java Persistence API (*JPA), *Java Message Service (*JMS), dentre outros em vários módulos distintos. A Figura 10 exibe um exemplo de anotações utilizando o Spring Framework.

```
@Controller
public class IndexController {

    @RequestMapping("/403")
    public String acessoNegado(Model model) {
        Pessoa auth = (Pessoa) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String name = auth.getNome();

        model.addAttribute("username", name);
        model.addAttribute("mensagem", "Acesso negado");
        return "403";
    }

}
```

Figura 10 - Exemplo de anotações do Spring Framework

3.1.5 Spring Data JPA

O Spring Data JPA é um dos projetos Spring Data com o propósito de fornecer suporte a repositórios JPA. Há tempos a implementação de acesso a dados

tem sido complexa, seja por repetição de código ou por necessidade de readequação das consultas para diferentes bancos de dados. O objetivo do Spring Data JPA é facilitar o acesso aos dados, uma vez que é necessário apenas que uma interface seja codificada para que os métodos padrões, como consultas por chave, persistência, atualização de dados, entre outros, sejam fornecidos automaticamente pelo *framework* (SPRING DATA JPA, 2015).

Dentre as funcionalidades, destacam-se:

1 – Suporte a predicados *Querydsl*[3];

2 – Suporte à paginação e execução de consultas dinâmicas;

3 – Suporte a entidades mapeadas em Ex*tensible Markup Language* (XML);

A Figura 11 exibe um exemplo de utilização do Spring Data JPA.

```
public interface ProdutoData extends JpaRepository<Produto, Long> {

    List<Produto> findByEstab(Estabelecimento estab);

    List<Produto> findByEstabAndGrupo(Estabelecimento estab, Grupo grupo);

    List<Produto> findByGrupo(Grupo grupo);

    List<Produto> findByDescricaoLike(String descricao);

    @Query("SELECT p FROM Produto p WHERE UPPER(p.grupo.descricao) like '%UPPER(:grupo)%' ")
    List<Produto> findByGrupoDescLike(@Param(value = "grupo") String grupo);

}
```

Figura 11 - Exemplo com Spring Data JPA

3.1.6 Spring Security

O Spring Security é um *framework* de autenticação e controle de acesso focado em oferecer autenticação e autorização em aplicações Java. Possui como principais funcionalidades proteção contra ataques em sessões, *clickjacking*[4], possibilidade de integração com o Spring Web MVC, dentre outros (SPRING

3 Ferramenta para criação de *queries type-safe.*

4 Armadilha preparada para que o usuário pense que está fazendo determinada ação, quando na verdade os cliques executados estão sendo utilizados para executar ações maliciosas.

SECURITY, 2015). A Figura 12 exibe a configuração de interceptação de URL com o Spring Security.

```
<security:http auto-config="false" use-expressions="true">
    <security:form-login login-page="/login/"
                         authentication-failure-url="/login/?error=bad_credentials"
                         default-target-url="/" />

    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/j_spring_security_check" access="permitAll"/>
    <security:intercept-url pattern="/resources/**" access="permitAll" />
    <security:intercept-url pattern="/login/" access="permitAll" />
    <security:intercept-url pattern="/logout" access="permitAll" />
    <security:intercept-url pattern="/menu/" access="permitAll" />
    <security:intercept-url pattern="/usuario/**" access="permitAll" />
    <security:intercept-url pattern="/carrinho/**" access="hasAnyRole('ROLE_ADMIN')" />
    <security:intercept-url pattern="/admin/**" access="hasAnyRole('ROLE_ADMIN')" />
    <security:intercept-url pattern="/lojista/**" access="hasAnyRole('ROLE_LOJISTA')"/>

    <security:access-denied-handler error-page="/403" />
    <security:logout logout-url="/logout" logout-success-url="/" invalidate-session="true"/>
</security:http>
```

Figura 12 - Configuração de interceptação de URL com Spring Security

3.1.7 Apache Maven

O *framework* Apache Maven é uma ferramenta de gerenciamento e compreensão de projetos baseado no conceito *Project Object Model* (POM). Inicialmente concebido apenas para gerenciar o *build* de projetos, possibilita a visualização de relatórios, documentação, cobertura de testes, gerenciamento de dependências, configuração de vários repositórios, dentre outros (MAVEN, 2015).

3.1.8 MySQL SGBD

O MySQL é o sistema de gerenciamento de banco de dados *Structured Query Language (*SQL) *open source* mais popular atualmente. Desenvolvido e distribuído pela Oracle, é um banco de dados relacional muito veloz, de fácil utilização e escalável. Possui suporte a muitos tipos de dados, como ***Float*** e ***Enum*** e diversas funcionalidades como as cláusulas ***Group By*** e ***Order By***, retorno de número de linhas afetadas em comandos de ***Delete***, ***Insert***, ***Update*** e ***Replaces***,

recuperação de informações das tabelas por meio do comando ***Show***, bem como o comando ***Explain*** com a função de exemplificar como a consulta é resolvida (MYSQL, 2015).

3.1.9 HTML 5

Linguagem de marcação utilizada para criação de documentos na *web*. Atualmente (versão 5) permite a criação de páginas com conteúdo semântico, além de padronizar o entendimento das seções do documento (Seção 2.5).

3.1.10 SB Admin 2

Tema para desenvolvimento de interfaces *web*, apresenta uma variedade de *plugins* personalizados para adicionar funcionalidades além do disponível no Boostrap UI (*User Interface*). Possui menu lateral e superior responsivos e com múltiplos níveis, painéis personalizados em três cores, *widget* para bate-papo, tabelas de dados com ordenação, filtro e paginação, *plugins* para gráficos, entre outros (SB ADMIN 2, 2015).

3.1.11 WildFly 8

Servidor de aplicações web mantido pela Jboss, possui uma abordagem de minimização de utilização de memória, oferecendo redução de dez vezes em tempo de inicialização quando comparado com outros servidores (WILDFLY, 2015). Implementa a versão Java EE 7 e possui suporte a vários projetos *open source* como a exemplo do Hibernate, RESTEasy e Mojarra (WILDFLY, 2015). A Figura 13 exibe o console de administração do WildFly.

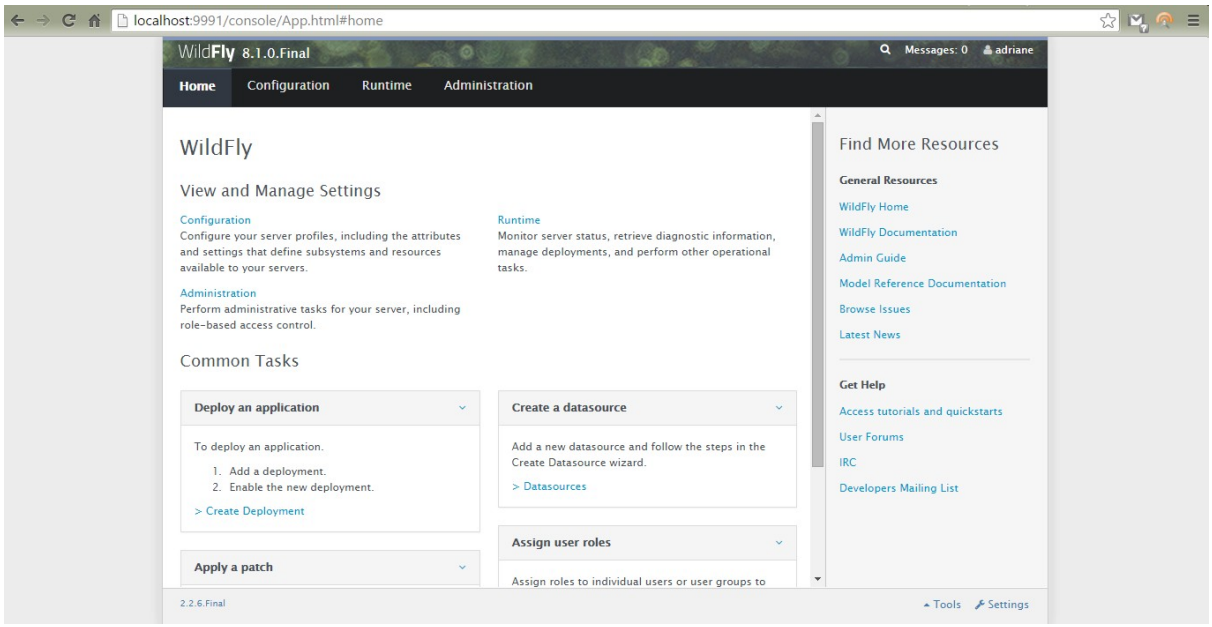

Figura 13 - Console de administração do WildFly

3.1.12 Hibernate

O Hibernate *Object Relational Mapping* (ORM) é uma implementação da especificação JPA que provê o mapeamento de base de dados relacionais. Contando com ampla documentação, o Hibernate pode ser utilizado em aplicações Java *Standard Edition (SE)*, Java *Enterprise Edition (EE)*, entre outros. A Figura 14 exibe o mapeamento objeto relacional com Hibernate.

```java
package br.com.utfpr.posjava.delivery.entity;

import ...

@Table
@Entity
public class Produto implements Serializable {

    @Id
    @Column
    @GeneratedValue(strategy = GenerationType.IDENTITY  )
    private Long id;

    @NotNull
    @NotEmpty
    @Column
    @Length(max = 60)
    private String descricao;

    @NotNull
    @Column
    @Digits(integer = 9, fraction = 2)
    private BigDecimal valor;

    @ManyToOne(optional = false)
    private Estabelecimento estab;

    @ManyToOne(optional = true)
    private Grupo grupo;
```

Figura 14 - Mapeamento Objeto Relacional com Hibernate

3.2 MÉTODOS

O método utilizado para o desenvolvimento do aplicativo engloba as fases de planejamento do sistema, análise, preparação do ambiente, implementação do módulo e testes.

a) Planejamento do sistema: inicialmente idealizou-se a visão geral do sistema. Em seguida ocorreu o levantamento bibliográfico visando encontrar os *frameworks* e ferramentas para desenvolvimento do sistema.

b) Análise, projeto e construção da *Unified Modeling Language (*UML): nessa fase foi realizada a análise do sistema, envolvendo a definição dos casos de uso, classes e atributos, operações e relacionamentos. Não houve necessidade de

desenvolver o projeto de banco de dados, bem como geração de *scripts* pois o projeto foi concebido com tecnologia que realiza tal gerenciamento.

c) Preparação do ambiente: nessa fase foi realizada a preparação, instalação e configuração do projeto para a implementação.

d) Implementação: o desenvolvimento do sistema foi realizado utilizando a IDE IntelliJ IDEA, com a construção da interface em HTML 5 e Bootstrap.

e) Realização de testes: os testes realizados foram manuais e executados pelo próprio desenvolvedor.

Considerando a utilização do *framework* Apache Maven, o gerenciamento de dependências dos projetos foi minimizado, sendo possível inclusive dividir o desenvolvimento do sistema em quatro projetos. A Figura 15 apresenta a concepção e organização dos projetos na IDE IntelliJ IDEA.

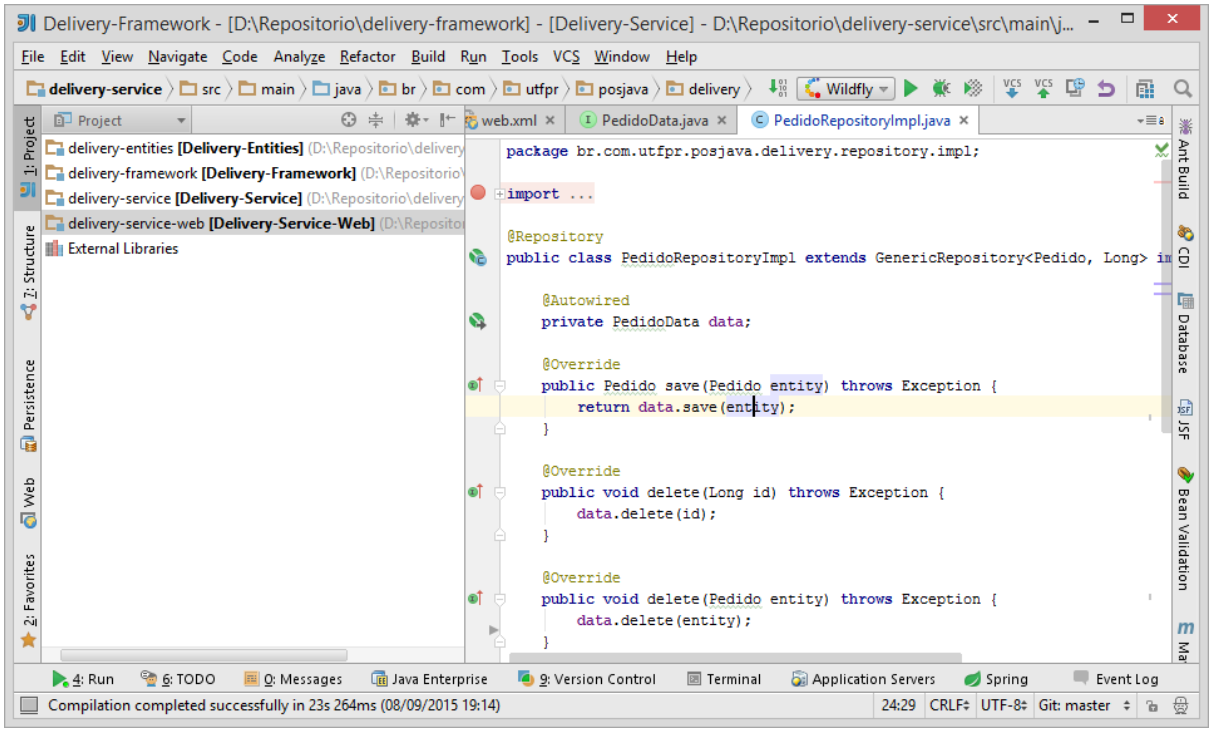

Figura 15 - Organização dos projetos na IDE IntelliJ IDEA

Haja vista que uma das tecnologias utilizadas no desenvolvimento, o Spring Web MVC, possui como premissa o uso do padrão MVC, o desenvolvimento foi organizado visando atender tal padrão. A camada *Model* fica distribuída conforme apresenta a Figura 16.

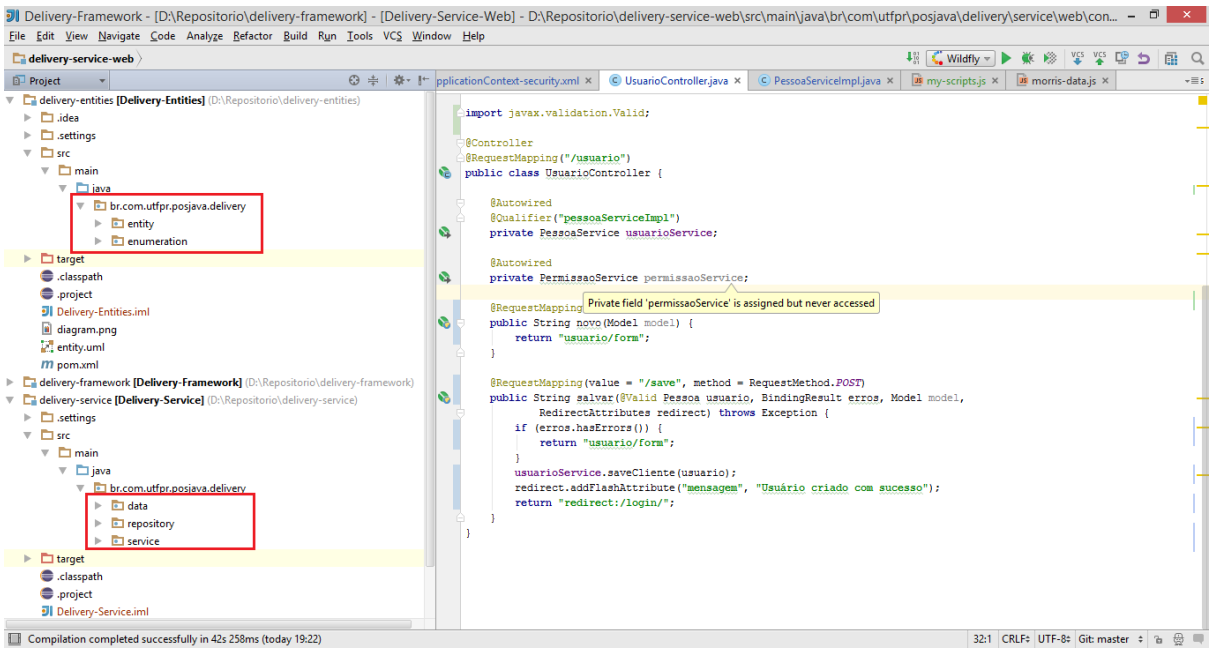

Figura 16 - Organização da camada Model

A camada *View* está ilustrada na Figura 17.

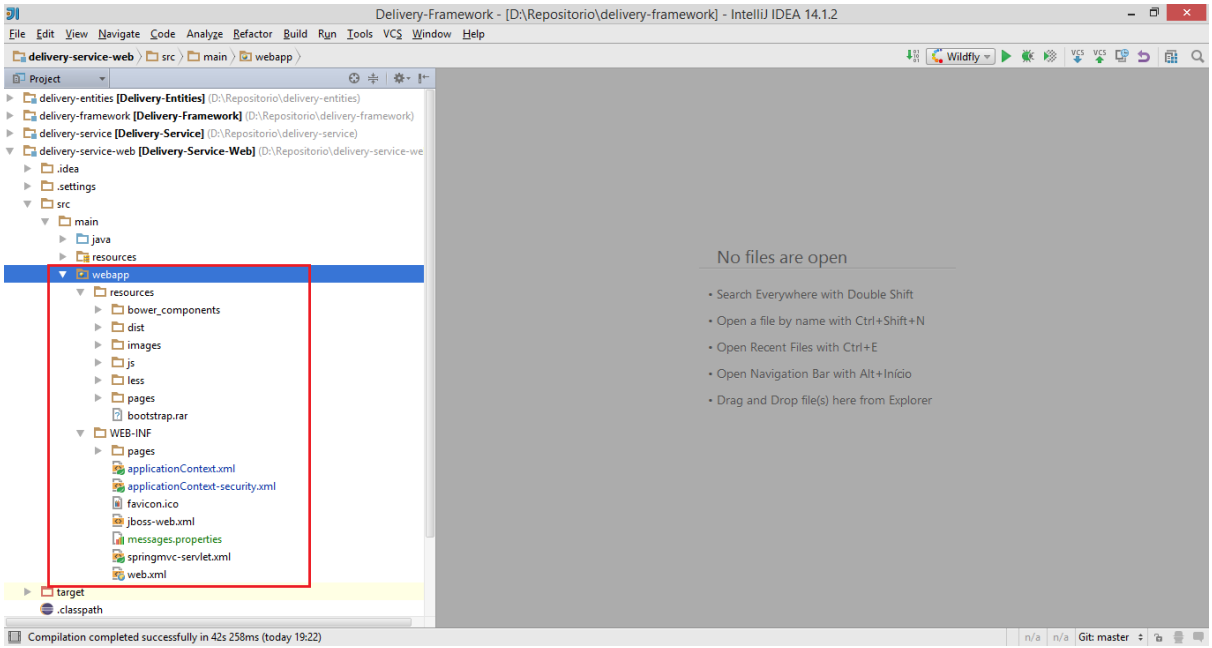

Figura 17 - Organização da camada View

A Figura 18 mostra a estrutura da camada de controle.

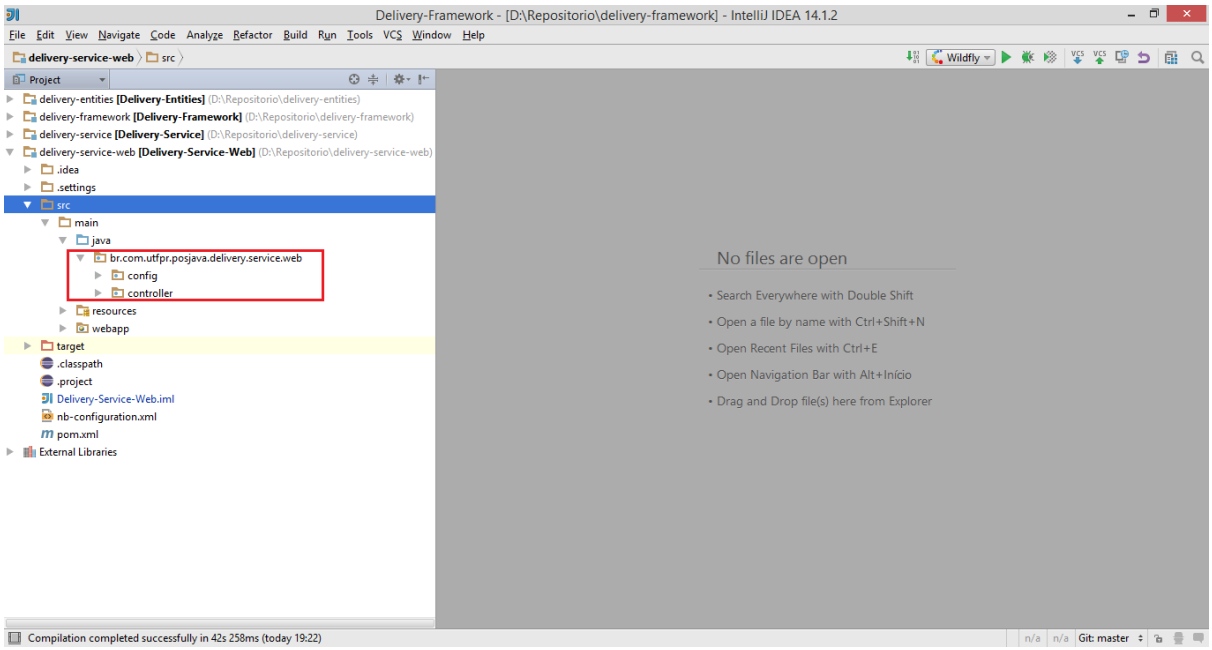

Figura 18 - Organização da camada *Controller*

4 RESULTADOS E DISCUSSÃO

Neste capítulo é apresentado o sistema para controle de entregas em domicílio implementado como resultado do desenvolvimento deste trabalho.

4.1 ESCOPO DO SISTEMA

O sistema de controle de entregas em domicílio tem por finalidade controlar o recebimento, a produção e a entrega de pedidos. Esse sistema permite acesso anônimo ou para usuários com o perfil de administrador, cliente ou lojista. Em um perfil administrador, é permitida a inclusão de produtos, grupos de produtos e usuários para o perfil lojista. Esse padrão de acesso compartilha com o perfil de lojista a possibilidade de receber e atualizar o *status* dos pedidos. Já em um acesso com padrão de cliente, é permitida a consulta de produtos, estabelecimentos, realização de novos pedidos e o histórico anterior.

4.2 MODELAGEM DO SISTEMA

Os requisitos identificados para o sistema são:

a) Controlar clientes:

- Quem são e sua localização.

b) Controlar os pedidos:

- Receber pedidos;
- Saber para qual cliente é o pedido;
- Atualizar o status de entrega do pedido.

Os requisitos estão representados como casos de uso na Figura 19. O sistema possuirá três atores:

a) Cliente: pessoas que realizam a solicitação de compra de um produto e para as quais são realizadas as entregas.

b) Lojista: ator com acesso a funcionalidades gerenciais do sistema. Possui acesso à consulta de itens, grupos e aprovação e acompanhamento de pedidos, mas não compartilha as funcionalidades do cliente.

c) Administrador: usuário semelhante ao Lojista, porém com acesso para cadastrar novos usuários, bem como a edição de grupos e produtos.

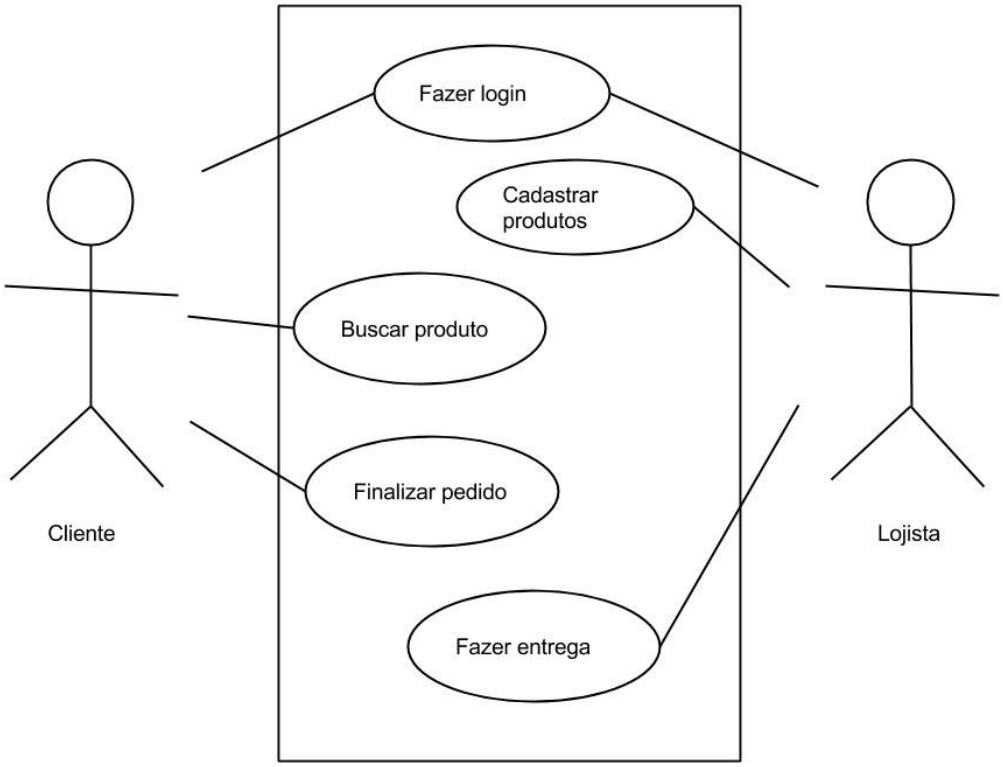

Figura 19 - Diagrama de Casos de Uso

Para atingir os requisitos e casos de uso propostos, foram definidas as classes e atributos conforme demonstra o diagrama de classes da Figura 20.

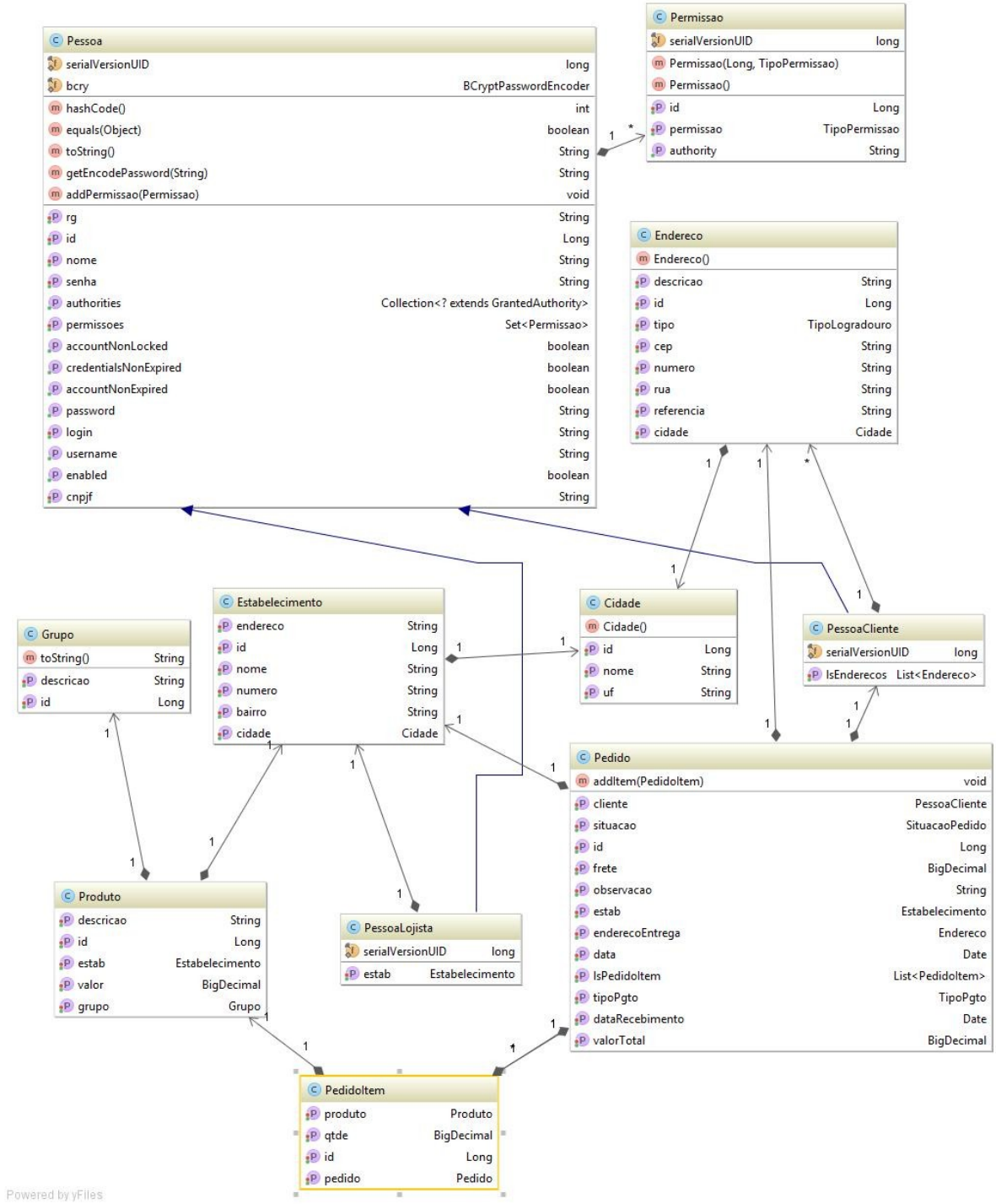

Figura 20 - Diagrama de Classes

Conforme apresenta o diagrama de classes na Figura 20, existem 11 classes de entidades envolvidas no desenvolvimento. Uma pessoa (usuário cliente ou lojista), além dos atributos RG, CPF, entre outros, possui vínculo com as permissões de acesso. Na especialização da classe pessoa para o usuário lojista

existe a limitação de acesso a estabelecimentos, enquanto para pessoa cliente existem a possibilidade de uso de vários endereços (vinculados a uma cidade). A distinção entre um usuário cliente ou usuário lojista é realizado pela coluna “TIPO”, configuração indicada no mapeamento da classe, conforme mostra a Figura 21.

```
@Entity
@Table(uniqueConstraints = { @UniqueConstraint(columnNames = { "LOGIN" }) })
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TIPO", discriminatorType = DiscriminatorType.STRING)
public class Pessoa implements Serializable, UserDetails {
```

Figura 21 - Mapeamento da classe Pessoa

Como as classes “PessoaCliente” e “PessoaLojista” herdam da classe “Pessoa”, é necessário informar no mapeamento qual a informação será salva na coluna “TIPO”, conforme demonstra a Figura 22.

```
@Entity
@DiscriminatorValue("C")
public class PessoaCliente extends Pessoa implements Serializable {
```

Figura 22 - Mapeamento da classe PessoaCliente

Para tornar possível a utilização do sistema pelo usuário é necessário realizar o cadastro do estabelecimento, bem como dos grupos e produtos, representados pelas classes “Estabelecimento”, “Grupo” e “Produto”. Por fim, a realização do pedido pelo cliente é representada pelas classes “Pedido” e “PedidoItem”, as quais contém todas as informações necessárias para a comercialização do produto.

Apesar de não existir desenvolvimento de *script* de banco de dados devido aos *frameworks* utilizados no desenvolvimento, na Figura 23 está o diagrama de entidades e relacionamentos (DER) do banco de dados seguindo a estrutura proposta pelos casos de uso, requisitos e diagrama de classes.

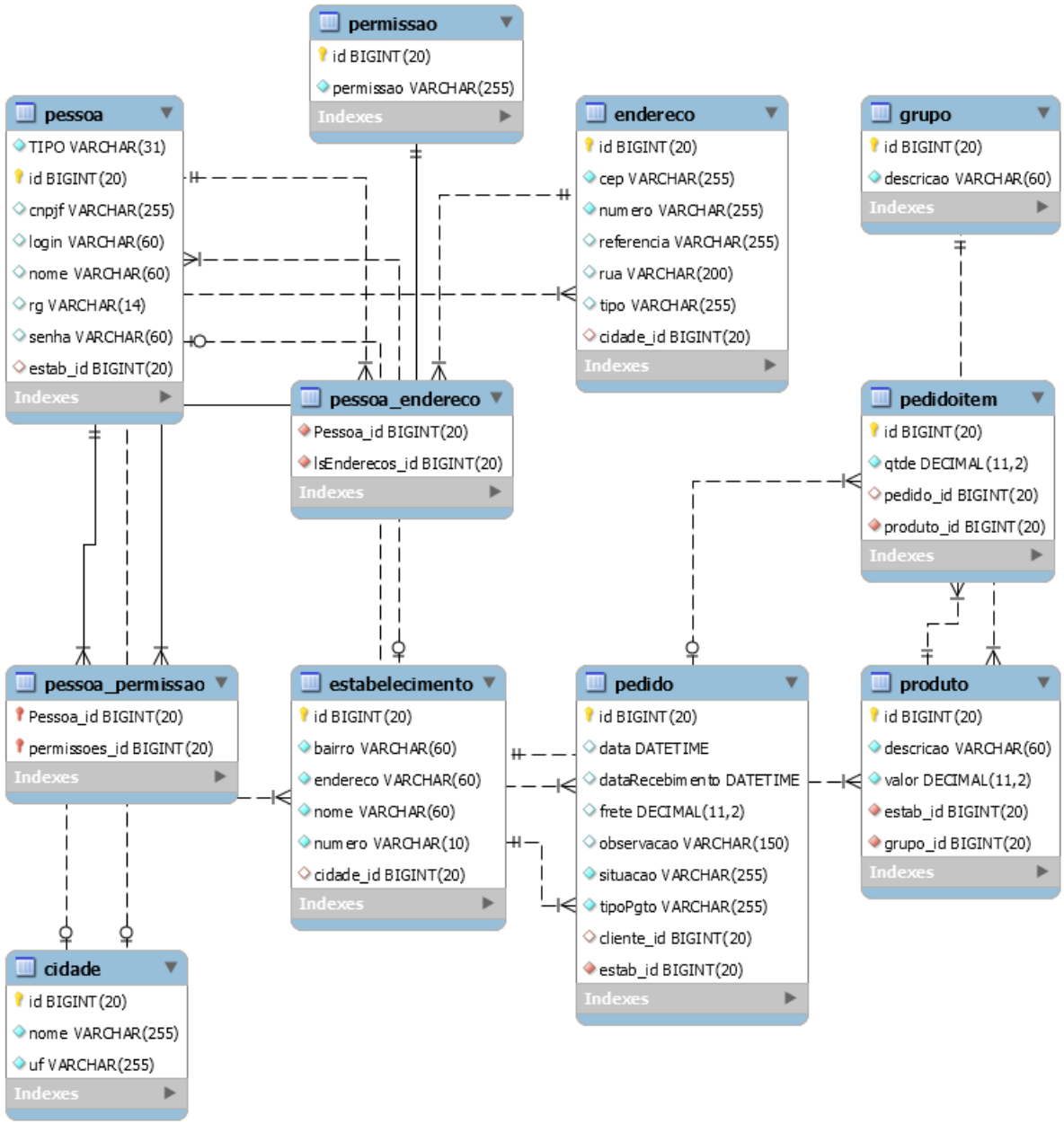

Figura 23 - Diagrama de entidades e relacionamentos de banco de dados

4.3 APRESENTAÇÃO DO SISTEMA

O acesso ao sistema é realizado pelo navegador no contexto “delivery” e na tela principal é apresentado o menu de navegação e a busca de produtos, localizada à esquerda, além do menu de ações, localizado à direita, conforme ilustrado na Figura 24.

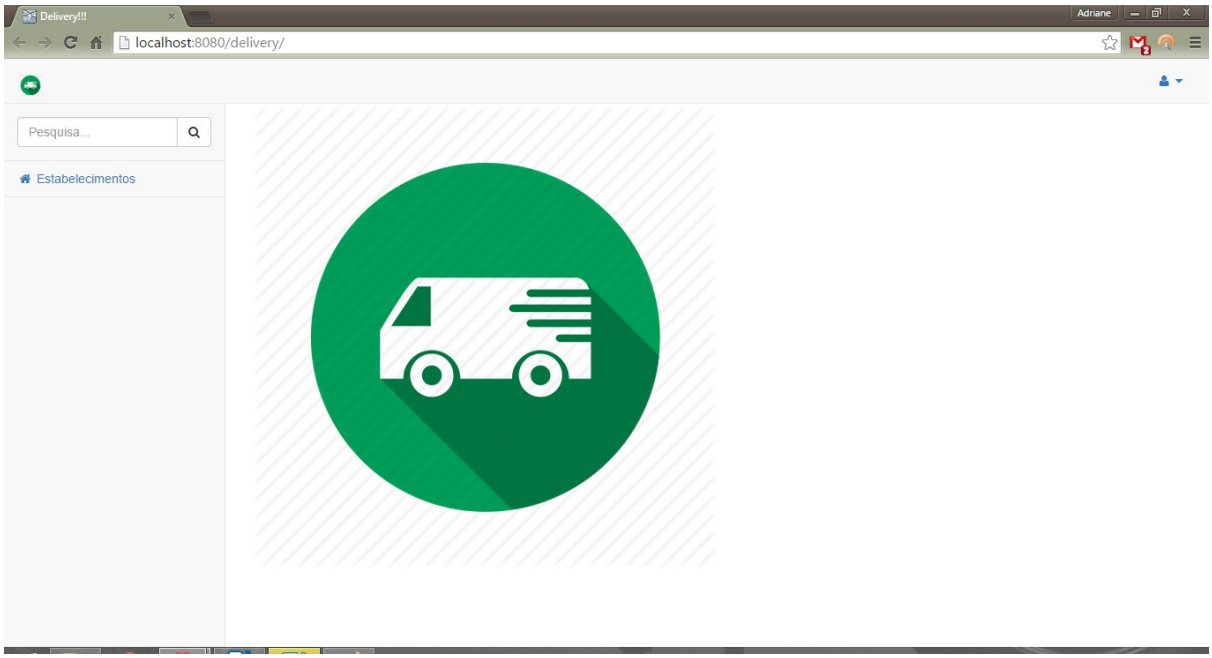

Figura 24 - Tela principal em maior resolução

O *framework* Boostrap permite criar sistemas responsivos e um exemplo é ilustrado na Figura 25 que exibe a tela principal do sistema em menor resolução. Na figura também é possível observar a alteração dos menus, bem como o tamanho da logo central apresentada.

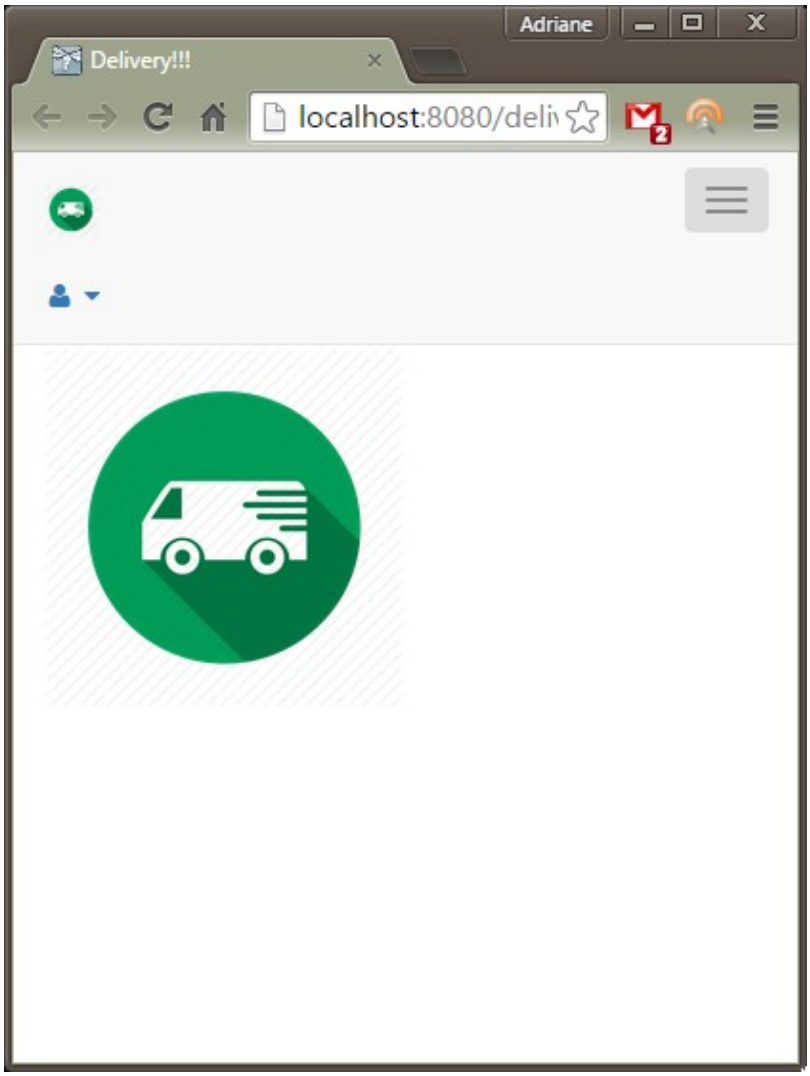

Figura 25 - Tela principal em menor resolução

O acesso anônimo (sem autenticação) permite somente a realização de consultas de estabelecimentos e produtos e a possibilidade de registrar-se conforme demonstra a Figura 26.

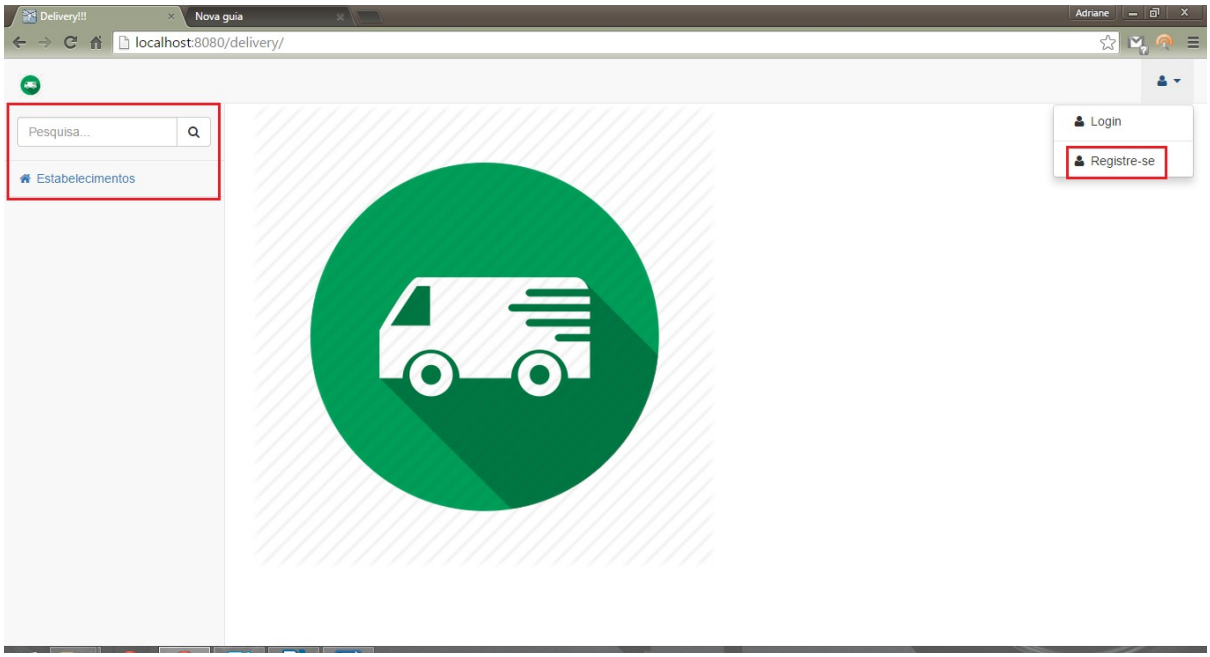

Figura 26 - Acesso anônimo

Para cadastrar-se o usuário deve preencher seu nome, CPF, RG, login e senha, como ilustra a Figura 27, sendo em seguida redirecionado para a tela de *login* no sistema.

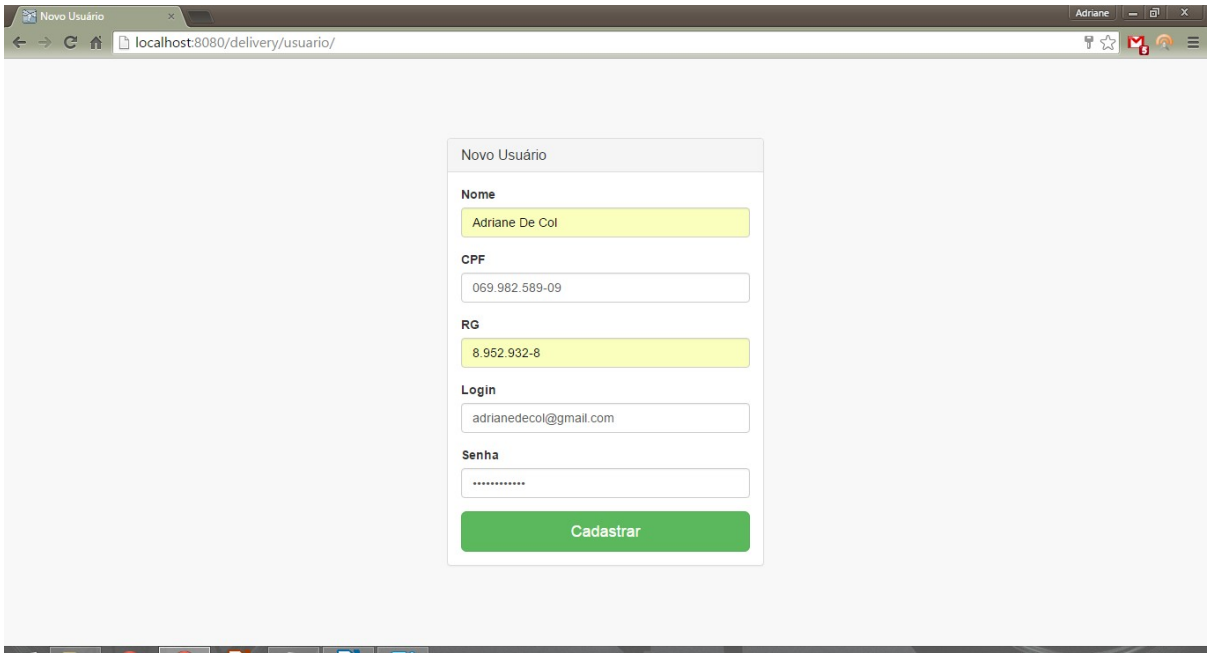

Figura 27 - Cadastro de Usuário

Ao contar com três perfis de acesso diferenciados, as opções disponíveis no sistema também sofrerão alteração. Ao realizar o *login* como cliente são apresentadas ao usuário somente opções de consulta de histórico de pedidos, carrinho de compras e busca de produtos, conforme ilustra a Figura 28.

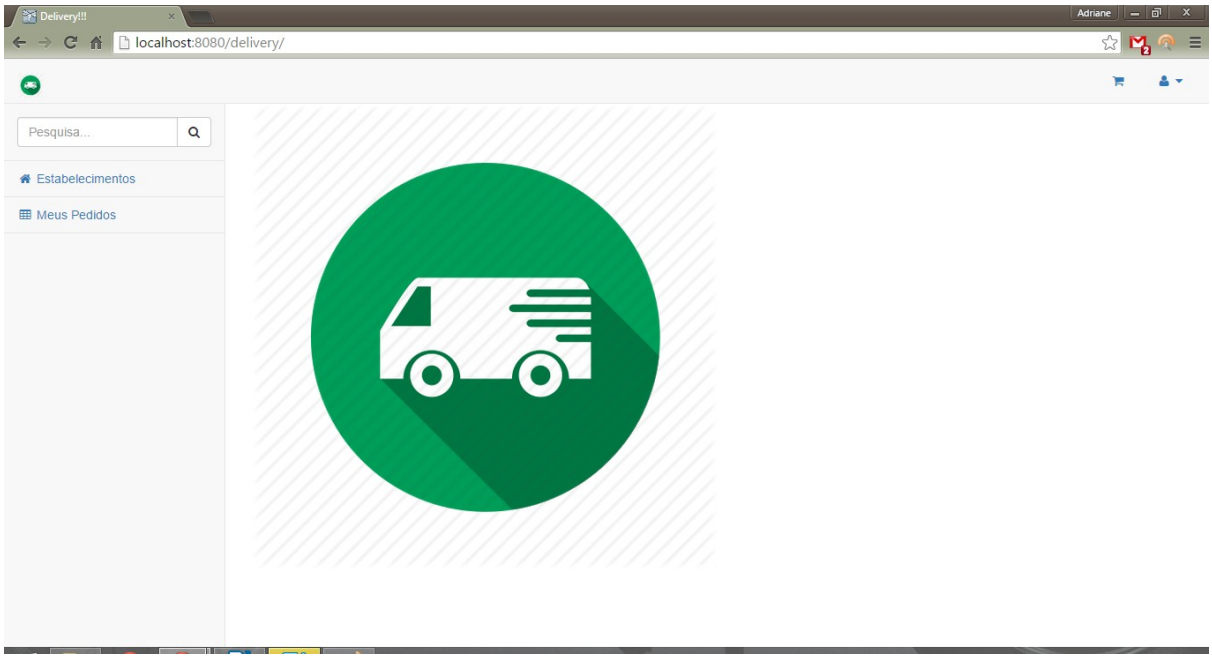

Figura 28 - Acesso como usuário cliente

A busca dos produtos pode ser realizada por meio do campo de pesquisa no canto superior esquerdo, onde a consulta será realizada em todos os estabelecimentos, ou ainda pelo menu Estabelecimentos, onde serão listados os produtos somente do estabelecimento em questão. A Figura 29 ilustra a tela resultante da pesquisa.

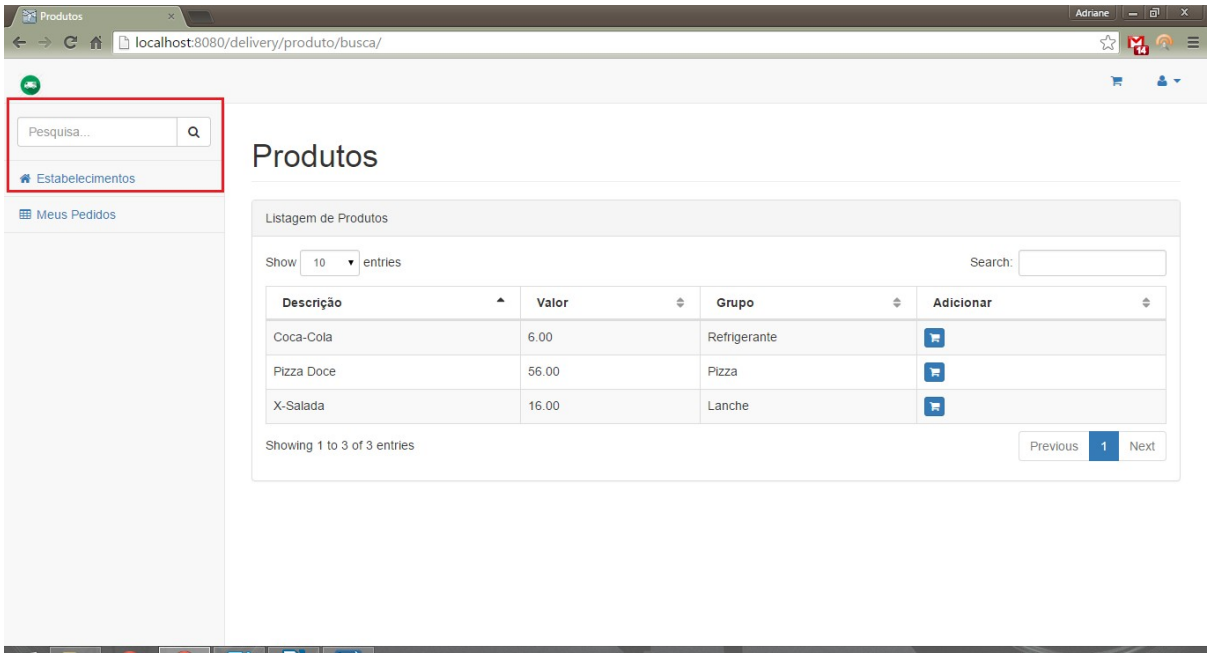

Figura 29 - Tela de busca de produtos

Ao solicitar a aquisição de um produto, o usuário é redirecionado à tela de carrinho de compras, onde são apresentados todos os seus pedidos pendentes. Nesta etapa é possível realizar a edição e conclusão da compra, conforme ilustra a Figura 30.

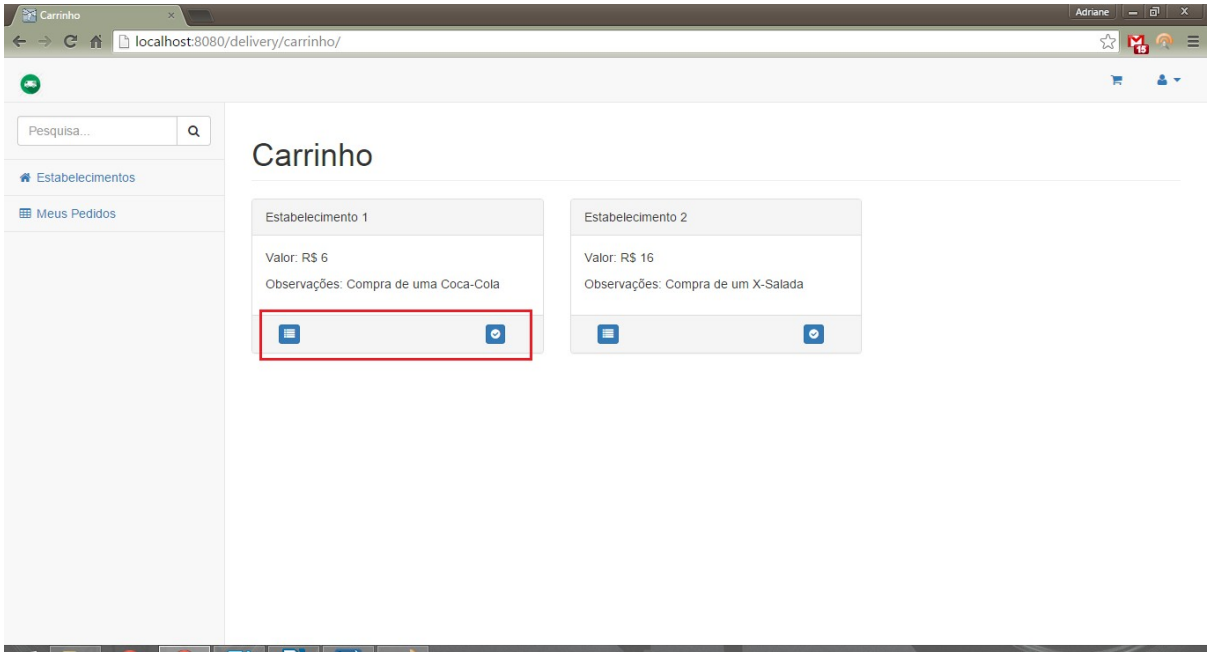

Figura 30 - Tela de carrinho de compras

A edição do pedido permite informar uma observação e realizar a seleção do endereço de entrega, conforme ilustra a Figura 31.

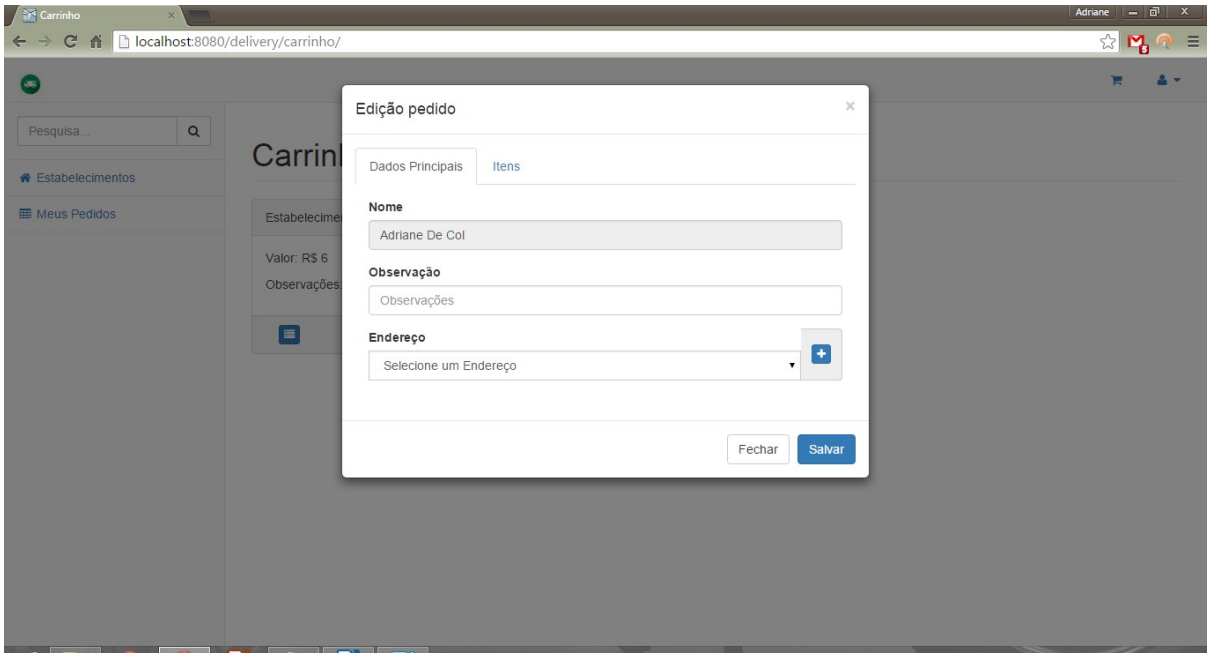

Figura 31 - Edição do pedido no carrinho de compras

Após o preenchimento dos dados necessários, o usuário pode finalizar o pedido, realizando o acompanhamento da solicitação pelo menu Meus Pedidos, conforme apresenta a Figura 32.

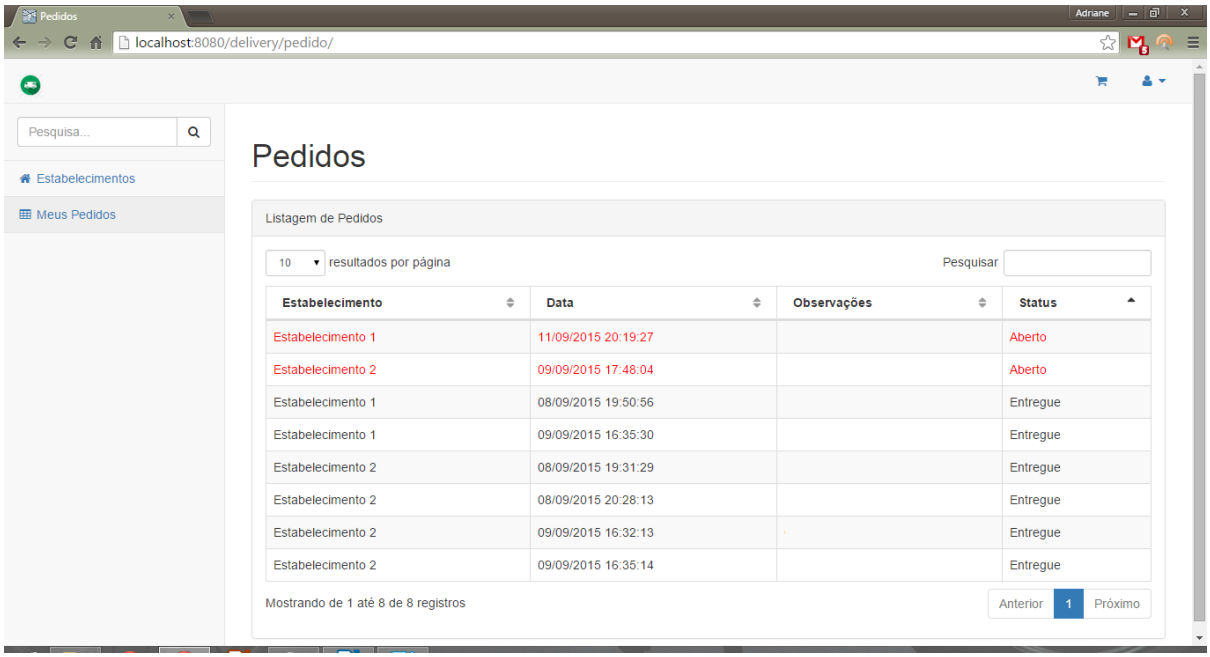

Figura 32 - Histórico de pedidos

O acesso com o perfil administrador permite a inclusão e edição de usuários (menu Usuários), grupos (menu Grupos) e produtos (menu Produtos), além do controle de pedidos pendentes conforme demonstra a Figura 33.

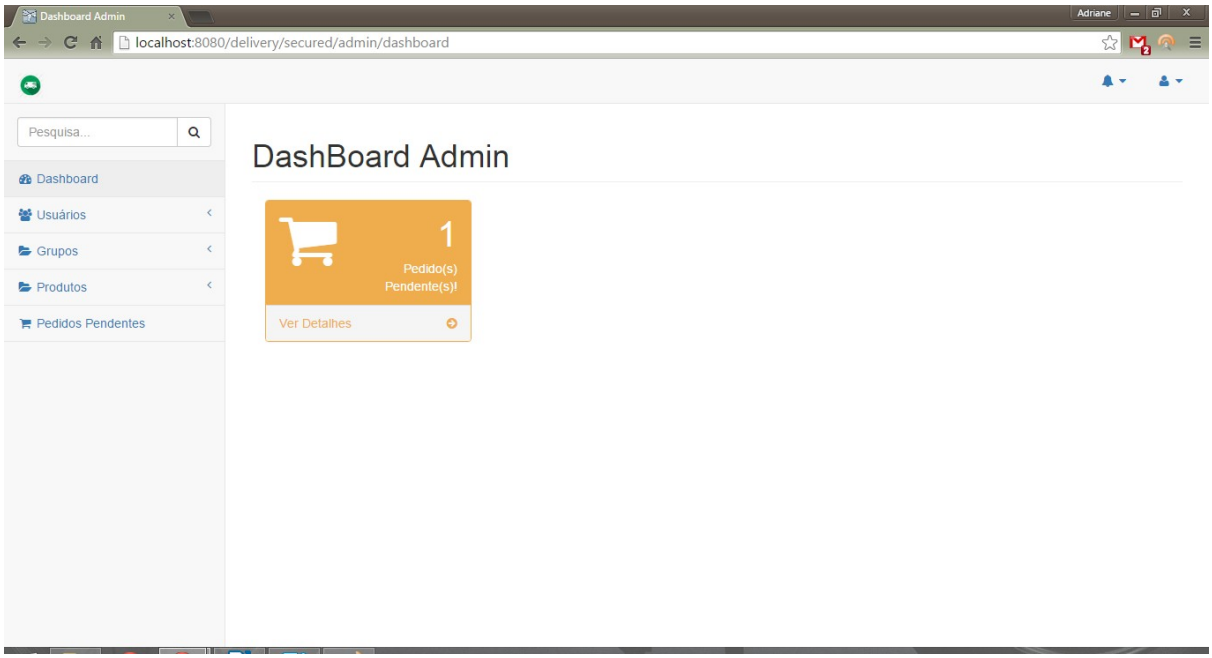

Figura 33 - Acesso como usuário administrador

Semelhante ao perfil administrador, o lojista tem o controle total dos pedidos, mas possui restrição de inclusão e alteração de grupos, produtos e acesso ao cadastro e consulta de outros usuários, conforme mostra a Figura 34.

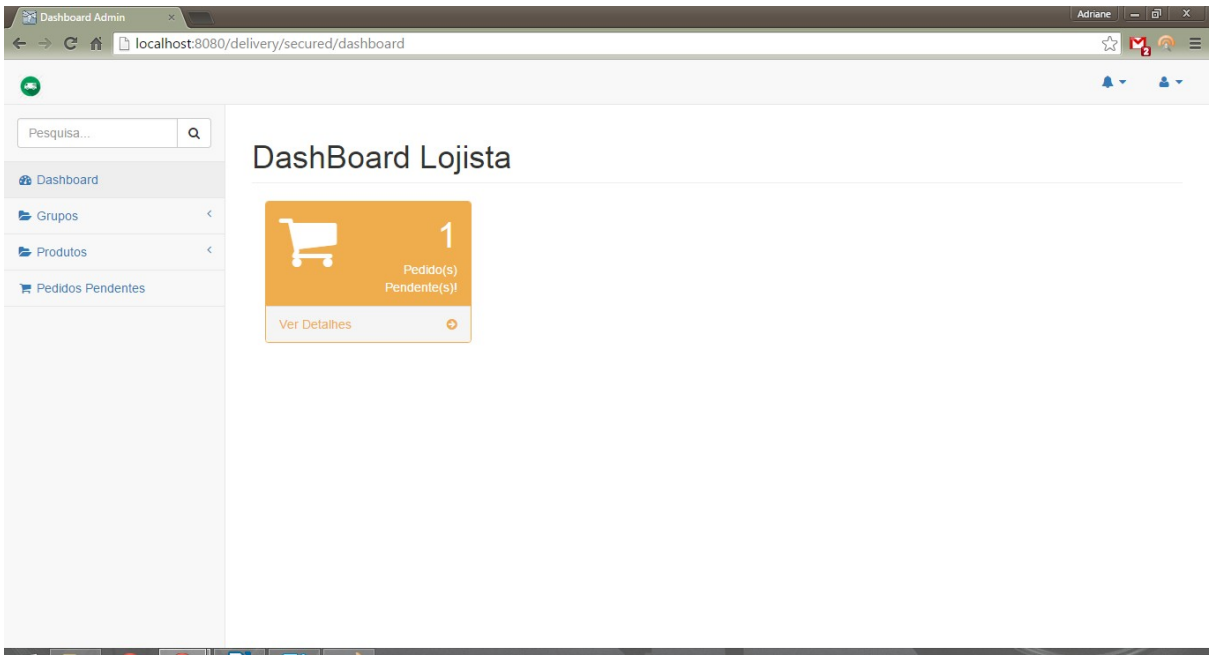

Figura 34 - Acesso como usuário lojista

O controle dos pedidos é realizado pelo menu Pedidos Pendentes, onde é realizado o acompanhamento e atualização da situação de preparo e entrega, conforme lista a Figura 35.

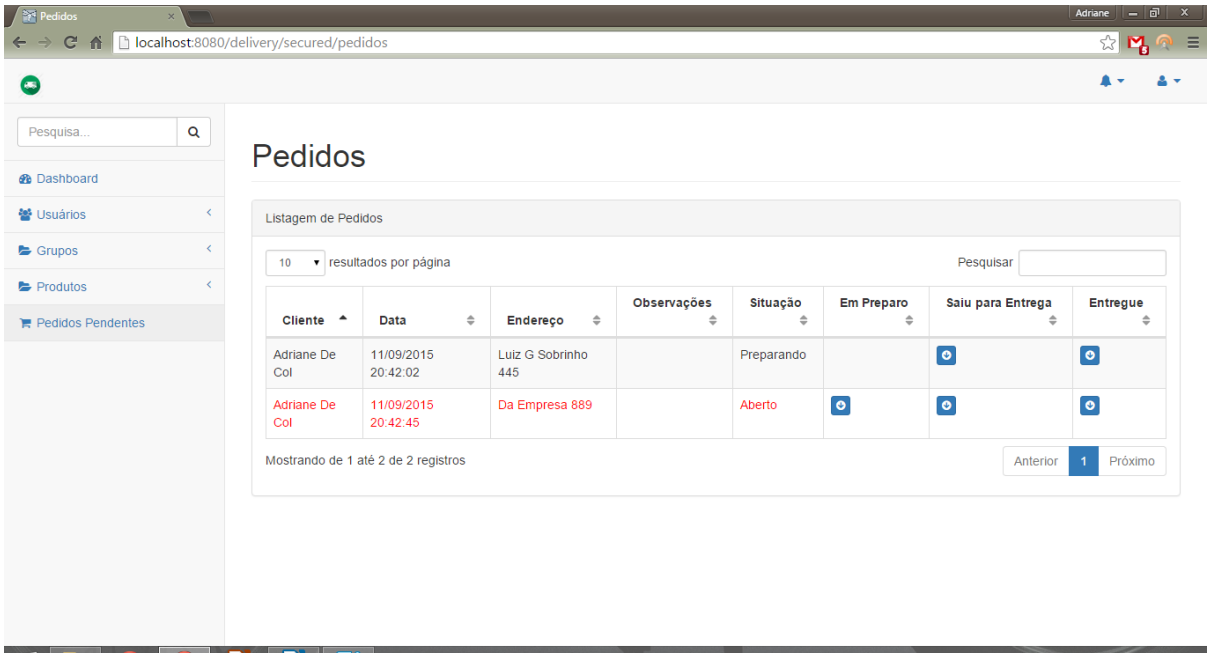

Figura 35 - Tela de Controle de Pedidos

4.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção são apresentados alguns códigos para exemplificar o desenvolvimento com Spring, HTML 5 e Bootstrap. A aplicação é um controle de entregas em domicílio.

4.4.1 Mapeamento entidade relacional

A primeira fase do desenvolvimento foi a realização do mapeamento entidade relacional. A Listagem 1 representa o mapeamento da tabela PRODUTO na classe “Produto”.

```
package br.com.utfpr.posjava.delivery.entity;

import java.io.Serializable;
import java.math.BigDecimal;

import javax.persistence.*;
import javax.validation.constraints.Digits;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;
```

```
@Table
@Entity
public class Produto implements Serializable {

    @Id
    @Column
    @GeneratedValue(strategy = GenerationType.IDENTITY  )
    private Long id;

    @NotEmpty
    @Column
    @Length(max = 60, min = 5, message = "{descricao.tamanho}")
    private String descricao;

    @NotNull(message = "{valor.obrigatorio}")
    @Column
    @Digits(integer = 9, fraction = 2)
    private BigDecimal valor;


    @ManyToOne(optional = false)
    private Estabelecimento estab;

    @ManyToOne(optional = true)
    @NotNull(message = "{grupo.obrigatorio}")
    private Grupo grupo;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }


    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }
    public BigDecimal getValor() {
        return valor;
    }

    public void setValor(BigDecimal valor) {
        this.valor = valor;
    }

    public Estabelecimento getEstab() {
        return estab;
    }

    public void setEstab(Estabelecimento estab) {
        this.estab = estab;
```

```
    }


   public Grupo getGrupo() {
       return grupo;
    }

    public void setGrupo(Grupo grupo) {
       this.grupo = grupo;
    }

}
```

Listagem 1 - Classe "Produto"

4.4.2 Acesso ao banco de dados

Após finalizar o mapeamento de todas as entidades é necessário criar o acesso aos dados. O acesso ao banco de dados é realizado por três camadas: serviço, repositório e acesso aos dados.

As classes de serviço possuem apenas validações e regras de negócio, porém necessitam da anotação @Service para que o Spring faça o seu gerenciamento, conforme demonstra a Listagem 2.

```
package br.com.utfpr.posjava.delivery.service.impl;

import java.util.List;
import br.com.utfpr.posjava.delivery.entity.Estabelecimento;
import br.com.utfpr.posjava.delivery.entity.PessoaLojista;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;
import br.com.utfpr.posjava.delivery.entity.Produto;
import br.com.utfpr.posjava.delivery.repository.ProdutoRepository;
import br.com.utfpr.posjava.delivery.service.ProdutoService;

@Service
public class ProdutoServiceImpl implements ProdutoService {

    @Autowired
    private ProdutoRepository repo;
    @Override
    public Produto save(Produto entity) throws Exception {
        PessoaLojista auth = (PessoaLojista) SecurityContextHolder.getContext().
                                getAuthentication().getPrincipal();
        entity.setEstab(auth.getEstab());
        return repo.save(entity);
    }
    @Override
    public void delete(Long id) throws Exception {
```

```
        repo.delete(id);
    }

    @Override
    public void delete(Produto entity) throws Exception {
        repo.delete(entity);
    }

    @Override
    public List<Produto> findAll() {
        return repo.findAll();
    }

    @Override
    public Produto findById(Long id) {
        return repo.findById(id);
    }

    @Override
    public List<Produto> findByEstab(Estabelecimento estab) {
        return repo.findByEstab(estab);
    }

    @Override
    public List<Produto> findByDescricaoContainingIgnoreCase(String descricao) {
        return repo.findByDescricaoContainingIgnoreCase(descricao);
    }

}
```

Listagem 2 - Classe de serviço "ProdutoServiceImpl"

Com as validações e regras de negócio já finalizadas, é necessário acesso aos dados. Para isso são utilizadas as classes de repositório, as quais necessitam da anotação @Repository, conforme mostra a Listagem 3.

```
package br.com.utfpr.posjava.delivery.repository.impl;

import java.util.List;
import br.com.utfpr.posjava.delivery.entity.Estabelecimento;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import br.com.utfpr.posjava.delivery.data.ProdutoData;
import br.com.utfpr.posjava.delivery.entity.Grupo;
import br.com.utfpr.posjava.delivery.entity.Produto;
import br.com.utfpr.posjava.delivery.framework.GenericRepository;
import br.com.utfpr.posjava.delivery.repository.ProdutoRepository;

@Repository
public class ProdutoRepositoryImpl extends GenericRepository<Produto, Long>
                                    implements ProdutoRepository {
    @Autowired
    private ProdutoData data;
```

```
    @Override
    public List<Produto> findByDescricaoLike(String descricao) {
        return data.findByDescricaoLike(descricao);
    }

    @Override
    public List<Produto> findByGrupoDescLike(String descGrupo) {
        return data.findByGrupoDescLike(descGrupo);
    }

    @Override
    public List<Produto> findByEstab(Estabelecimento estab) {
        return data.findByEstab(estab);
    }

    @Override
    public List<Produto> findByDescricaoContainingIgnoreCase(String descricao) {
        return data.findByDescricaoContainingIgnoreCase(descricao);
    }

    @Override
    public Produto save(Produto entity) throws Exception {
        return data.save(entity);
    }

    @Override
    public void delete(Long id) throws Exception {
        data.delete(id);
    }

    @Override
    public void delete(Produto entity) throws Exception {
        data.delete(entity);
    }

    @Override
    public List<Produto> findAll() {
        return data.findAll();
    }

    @Override
    public Produto findById(Long id) {
        return data.findOne(id);
    }

}
```

Listagem 3 - Classe de repositório "ProdutoRepositoryImpl"

As classes de repositório somente indicam ao Spring que estas desempenham essa função, porém as mesmas não possuem acesso aos dados. Como último passo é necessário criar as interfaces Data, as quais estendem JpaRepository e fornecem acesso aos dados, conforme mostra a Listagem 4.

```
package br.com.utfpr.posjava.delivery.data;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

import br.com.utfpr.posjava.delivery.entity.Estabelecimento;
import br.com.utfpr.posjava.delivery.entity.Grupo;
import br.com.utfpr.posjava.delivery.entity.Produto;

@Transactional(propagation=Propagation.REQUIRED)
public interface ProdutoData extends JpaRepository<Produto, Long> {

    List<Produto> findByEstab(Estabelecimento estab);

    List<Produto> findByEstabAndGrupo(Estabelecimento estab, Grupo grupo);

    List<Produto> findByGrupo(Grupo grupo);

    List<Produto> findByDescricaoLike(String descricao);

    @Query("SELECT p FROM Produto p WHERE UPPER(p.grupo.descricao) like
            '%UPPER(:grupo)%' ")
    List<Produto> findByGrupoDescLike(@Param(value = "grupo") String grupo);

    List<Produto> findByDescricaoContainingIgnoreCase(String descricao);

}
```

Listagem 4 - Classe de acesso a dados “ProdutoData”

Finalizada a criação do mapeamento e das classes de acesso ao banco de dados, é necessário criar e configurar o arquivo “persistence.xml”, o qual fornece dados ao JPA de classes que serão gerenciadas por ele, do provedor de dados, bem como propriedades específicas da conexão, como dialeto, controle de transações, entre outros. A Listagem 5 apresenta essas configurações.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="persistenceJpa" transaction-type="JTA">

      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:jboss/datasources/Delivery</jta-data-source>
      <class>br.com.utfpr.posjava.delivery.entity.Cidade</class>
      <class>br.com.utfpr.posjava.delivery.entity.Estabelecimento</class>
```

```
      <class>br.com.utfpr.posjava.delivery.entity.Favorito</class>
      <class>br.com.utfpr.posjava.delivery.entity.Grupo</class>
      <class>br.com.utfpr.posjava.delivery.entity.Pedido</class>
      <class>br.com.utfpr.posjava.delivery.entity.PedidoItem</class>
      <class>br.com.utfpr.posjava.delivery.entity.Endereco</class>
      <class>br.com.utfpr.posjava.delivery.entity.Pessoa</class>
      <class>br.com.utfpr.posjava.delivery.entity.PessoaCliente</class>
      <class>br.com.utfpr.posjava.delivery.entity.PessoaLojista</class>
      <class>br.com.utfpr.posjava.delivery.entity.Permissao</class>
      <class>br.com.utfpr.posjava.delivery.entity.Produto</class>

      <properties>
         <property name="hibernate.transaction.jta.platform"
                   value="org.hibernate.service.jta.platform.internal.
                          JBossAppServerJtaPlatform" />
         <property name="hibernate.hbm2ddl.auto" value="update" />
         <property name="hibernate.dialect"
                   value="org.hibernate.dialect.MySQLDialect" />
         <property name="hibernate.show_sql" value="true" />
         <property name="hibernate.jdbc.batch_size" value="0" />
         <property name="hibernate.connection.charSet" value="WE8ISO8859P1" />
         <property name="hibernate.connection.useUnicode" value="true" />
      </properties>
   </persistence-unit>
</persistence>
```

Listagem 5 - Arquivo "persistence.xml"

O *data source* denominado "Delivery", conforme mostra a Listagem 5, é o responsável pela conexão com o banco de dados e deve ser declarado e configurado no servidor de aplicações WildFly, como representa a Figura 36.

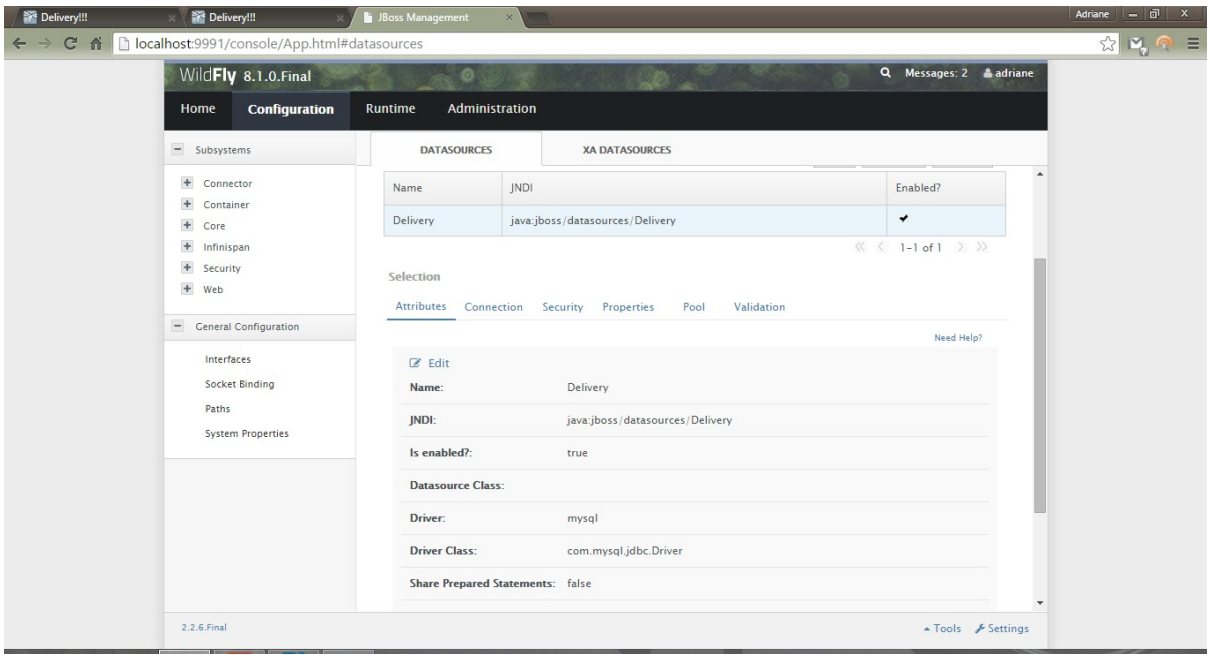

Figura 36 - Configuração do *Data Source* "Delivery" no WildFly

4.4.3 Configurando o Spring

A configuração do Spring é realizada no arquivo “applicationContext.xml”. Neste arquivo é apontado qual é o pacote base da aplicação, bem como indicado que o *pool* de conexões com o banco de dados será gerenciado pelo servidor de aplicações, dentre outros. A Listagem 6 mostra o arquivo XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:p="http://www.springframework.org/schema/p"
          xmlns:aop="http://www.springframework.org/schema/aop"
          xmlns:context="http://www.springframework.org/schema/context"
          xmlns:jpa="http://www.springframework.org/schema/data/jpa"
          xmlns:jee="http://www.springframework.org/schema/jee"
          xmlns:task="http://www.springframework.org/schema/task"
          xmlns:tx="http://www.springframework.org/schema/tx"
          xmlns:mvc="http://www.springframework.org/schema/mvc"
         xsi:schemaLocation="
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
      http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-
jee-3.0.xsd
      http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-
3.0.xsd
      http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task-3.0.xsd
      http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
      http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-
3.1.xsd
      http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">
        <context:annotation-config />
        <context:component-scan base-package="br.com.utfpr.posjava.delivery.*" />
        <bean id="entityManagerFactory"
              class="org.springframework.orm.jpa.
                    LocalContainerEntityManagerFactoryBean">
          <property name="persistenceUnitName" value="persistenceJpa" />
          <property name="jpaVendorAdapter">
                <bean class="org.springframework.orm.jpa.vendor.
                            HibernateJpaVendorAdapter">
                      <property name="generateDdl" value="false" />
                </bean>
          </property>
          <property name="jpaPropertyMap">
                <map>
```

```
                                    <entry key="hibernate.connection.autocommit"
                                        value="false" />
                                    <entry key="hibernate.connection.release_mode"
                                        value="after_transaction" />
                                </map>
                        </property>
                </bean>

                <bean id="transactionManager" class="org.springframework.orm.jpa.
                                                        JpaTransactionManager">
                        <property name="entityManagerFactory" ref="entityManagerFactory" />
                </bean>

                <tx:jta-transaction-manager />
                <tx:annotation-driven transaction-manager="transactionManager"/>

                <jpa:repositories base-package="br.com.utfpr.posjava.delivery.data"
                        entity-manager-factory-ref="entityManagerFactory"
                        transaction-manager-ref="transactionManager" />

                <bean class="org.springframework.orm.jpa.support.
                                PersistenceAnnotationBeanPostProcessor" />

                <bean class="org.springframework.dao.annotation.
                                PersistenceExceptionTranslationPostProcessor" />

                <!-- Configuração do arquivo de mensagens -->
                <bean id="messageSource" class="org.springframework.context.support.
                                                ResourceBundleMessageSource">
                        <property name="basename" value="ValidationMessages"/>
                </bean>

</beans>
```

Listagem 6 - Arquivo “applicationContext.xml”

Todo o processo de autenticação e autorização do sistema, ou seja, realização do login e controle de permissões de acesso é configurado no arquivo “applicationContext-security.xml”, conforme demonstra a Listagem 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:security="http://www.springframework.org/schema/security"
                xmlns:beans="http://www.springframework.org/schema/beans"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.springframework.org/schema/beans
                                http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                                http://www.springframework.org/schema/security
                                http://www.springframework.org/schema/security/spring-security-3.2.xsd">

   <security:http auto-config="false" use-expressions="true">
      <security:form-login login-page="/login/"
                        authentication-failure-url="/login/?error=bad_credentials"
                        default-target-url="/" />

      <security:intercept-url pattern="/" access="permitAll" />
      <security:intercept-url pattern="/j_spring_security_check" access="permitAll"/>
```

```
    <security:intercept-url pattern="/resources/**" access="permitAll" />
    <security:intercept-url pattern="/login/" access="permitAll" />
    <security:intercept-url pattern="/logout" access="permitAll" />
    <security:intercept-url pattern="/menu/" access="permitAll" />
    <security:intercept-url pattern="/usuario/**" access="permitAll" />
    <security:intercept-url pattern="/estabelecimento/**" access="permitAll" />
    <security:intercept-url pattern="/produto/**" access="permitAll" />
    <security:intercept-url pattern="/carrinho/**" access="hasAnyRole('ROLE_USER')" />
    <security:intercept-url pattern="/pedido/**" access="hasAnyRole('ROLE_USER')" />
    <security:intercept-url pattern="/secured/admin/**" access="hasAnyRole('ROLE_ADMIN')"/>
    <security:intercept-url pattern="/secured/**" access="hasAnyRole('ROLE_LOJISTA',
                                                        'ROLE_ADMIN')"/>

    <security:access-denied-handler error-page="/403" />
    <security:logout logout-url="/logout" logout-success-url="/" invalidate-session="true"/>
  </security:http>

  <!-- Classe responsável pela autenticação do usuário -->
  <beans:bean id="userDetailsService"
     class="br.com.utfpr.posjava.delivery.service.impl.PessoaServiceImpl">
  </beans:bean>

  <security:authentication-manager>
     <security:authentication-provider user-service-ref="userDetailsService">
        <security:password-encoder hash="bcrypt"/>
     </security:authentication-provider>
  </security:authentication-manager>

  <beans:bean id="webSecurityExpressionHandler"
           class="org.springframework.security.web.access.expression.
                    DefaultWebSecurityExpressionHandler" />

</beans:beans>
```

Listagem 7 - Arquivo "applicationContext-security.xml"

Para que o Spring Security realize o tratamento de autenticação e autorização, é necessário configurar as classes responsáveis por estes papéis. Uma vez que a classe "Pessoa" representa o usuário, é necessário que implemente a interface "UserDetails", conforme demonstra a Listagem 8.

```
package br.com.utfpr.posjava.delivery.entity;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Objects;
import java.util.Set;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
```

```
import org.apache.commons.lang3.StringUtils;
import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;
import org.hibernate.validator.constraints.br.CPF;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import br.com.utfpr.posjava.delivery.enumeration.TipoPessoa;

/**
 *
 * @author Adriane
 */
@Entity
@Table(uniqueConstraints = { @UniqueConstraint(columnNames = { "LOGIN" }) })
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TIPO", discriminatorType = DiscriminatorType.STRING)
public class Pessoa implements Serializable, UserDetails {

    private static final long serialVersionUID = 3302414787806026613L;

    private static final BCryptPasswordEncoder bcry = new BCryptPasswordEncoder();

    @Id
    @Column
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column
    @Length(max = 60, min = 10, message = "{nome.tamanho}")
    private String nome;

    @Column
    @CPF(message = "{cpf.invalido}")
    private String cnpjf;

    @Column
    @Length(max = 14, min = 5, message = "{rg.tamanho}")
    private String rg;

    @Column
    @Length(max = 60, min = 6, message = "{login.tamanho}")
    @Email(message = "{email.invalido}")
    private String login;

    @Column
    @Length(max = 60, min = 6, message = "{senha.tamanho}")
    private String senha;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
```

```
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getCnpjf() {
    return cnpjf;
}

public void setCnpjf(String cnpjf) {
    this.cnpjf = cnpjf;
}

public String getRg() {
    return rg;
}

public void setRg(String rg) {
    this.rg = rg;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

@ManyToMany(cascade = CascadeType.MERGE, fetch=FetchType.EAGER)
private Set<Permissao> permissoes;

@Override
public int hashCode() {
    int hash = 3;
    hash = 41 * hash + Objects.hashCode(this.id);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
```

```
            return false;
        }
        final Pessoa other = (Pessoa) obj;
        if (!Objects.equals(this.id, other.id)) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Pessoa{" + "id=" + id + ", nome=" + nome + ", cnpjf=" + cnpjf
                + ", rg=" + rg + ", login=" + login + ", senha=" + senha + '}';
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        List<GrantedAuthority> auto = new ArrayList<GrantedAuthority>();
        auto.addAll(getPermissoes());
        return auto;
    }

    @Override
    public String getPassword() {
        return this.senha;
    }

    @Override
    public String getUsername() {
        return this.login;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    public Set<Permissao> getPermissoes() {
        return permissoes;
    }

    public void setPermissoes(Set<Permissao> permissoes) {
        this.permissoes = permissoes;
```

```
    }

    public String getEncodePassword(String pass) {
        if (StringUtils.isNotEmpty(pass)) {
            return bcry.encode(pass);
        }
        return pass;
    }

    public void addPermissao(Permissao permissao) {
        if (permissoes == null) {
            permissoes = new HashSet<Permissao>();
        }
        permissoes.add(permissao);
    }

}
```

Listagem 8 - Classe "Pessoa"

A classe "PessoaServiceImpl" é responsável pela autorização e autenticação de usuários e deve implementar a interface "UserDetailsService"[5], conforme mostra a Listagem 9.

```
package br.com.utfpr.posjava.delivery.service.impl;

import java.util.List;
import br.com.utfpr.posjava.delivery.entity.Permissao;
import br.com.utfpr.posjava.delivery.enumeration.TipoPermissao;
import br.com.utfpr.posjava.delivery.enumeration.TipoPessoa;
import br.com.utfpr.posjava.delivery.service.PermissaoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import br.com.utfpr.posjava.delivery.entity.Pessoa;
import br.com.utfpr.posjava.delivery.repository.PessoaRepository;
import br.com.utfpr.posjava.delivery.service.PessoaService;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Service
public class PessoaServiceImpl implements PessoaService {

    @Autowired
    private PessoaRepository repo;

    @Override
    public Pessoa save(Pessoa entity) throws Exception {
        return repo.save(entity);
```

5 A classe "PessoaServiceImpl" já implementa uma interface, por isso a declaração "extends UserDetailsService" é realizada na interface "PessoaService".

```
    }

    @Override
    public void delete(Long id) throws Exception {
        repo.delete(id);

    }

    @Override
    public void delete(Pessoa entity) throws Exception {
        repo.delete(entity);

    }

    @Override
    public List<Pessoa> findAll() {
        return repo.findAll();
    }

    @Override
    public Pessoa findById(Long id) {
        return repo.findById(id);
    }

    @Override
    public UserDetails loadUserByUsername(String login)
            throws UsernameNotFoundException {
        Pessoa usuario = repo.findByLogin(login);
        if(usuario == null){
            throw new UsernameNotFoundException("Login invalido");
        }
        return usuario;
    }
}
```

Listagem 9 - Classe "PessoaServiceImpl"

A indicação do local das páginas HTML do sistema, bem como seu formato é apresentado no arquivo "springmvc-servlet.xml", representado na Listagem 10.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-3.0.xsd
                http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

        <context:component-scan base-package="br.com.utfpr.posjava.delivery.*" />

        <bean class="org.springframework.web.servlet.view.
```

```
            InternalResourceViewResolver">
                <property name="prefix" value="/WEB-INF/pages/" />
                <property name="suffix" value=".jsp" />
        </bean>

        <mvc:resources mapping="/resources/**" location="/resources/"/>

        <mvc:annotation-driven />
</beans>
```

Listagem 10 - Arquivo "springmvc-servlet.xml"

Por fim é necessário a configuração do arquivo "web.xml", onde será apontado os arquivos de configuração de contexto "applicationContext.xml" e "applicationContext-security.xml", bem como a URL base para interceptação do Spring Security, o servlet Dispatcher, dentre outros, conforme mostra a Listagem 11.

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

   <context-param>
      <param-name>webAppRootKey</param-name>
      <param-value>webapp.delivery.serviceweb</param-value>
   </context-param>

   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
         /WEB-INF/applicationContext.xml
         /WEB-INF/applicationContext-security.xml
      </param-value>
   </context-param>

   <filter>
         <filter-name>springSecurityFilterChain</filter-name>
     <filter-class> org.springframework.web.filter.DelegatingFilterProxy
     </filter-class>
   </filter>

   <filter-mapping>
     <filter-name>springSecurityFilterChain</filter-name>
     <url-pattern>/*</url-pattern>
   </filter-mapping>

   <listener>
      <listener-class>
           org.springframework.web.context.ContextLoaderListener
      </listener-class>
   </listener>

   <servlet>
```

```
            <servlet-name>springmvc</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
         </servlet-class>
            <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
            <servlet-name>springmvc</servlet-name>
            <url-pattern>/</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

</web-app>
```

Listagem 11 - Arquivo "web.xml"

4.4.5 Classes Controller e páginas HTML

Após concluir o mapeamento, acesso aos dados e configuração do Spring é possível realizar a implementação das classes de controle, as quais são responsáveis por responder às requisições das páginas HTML. O primeiro passo nessas classes é adicionar a anotação @Controller, em seguida é necessário criar os métodos e as URL's que estes responderão na anotação @RequestMapping. A Listagem 12 contém o *controller*[6] de produtos.

```
package br.com.utfpr.posjava.delivery.service.web.controller;

import br.com.utfpr.posjava.delivery.entity.Estabelecimento;
import br.com.utfpr.posjava.delivery.entity.Grupo;
import br.com.utfpr.posjava.delivery.entity.PessoaLojista;
import br.com.utfpr.posjava.delivery.entity.Produto;
import br.com.utfpr.posjava.delivery.service.EstabelecimentoService;
import br.com.utfpr.posjava.delivery.service.GrupoService;
import br.com.utfpr.posjava.delivery.service.ProdutoService;
import br.com.utfpr.posjava.delivery.service.web.editor.GrupoEditor;
import org.springframework.beans.factory.annotation.Autowired;
```

6 É comum a utilização da anotação @Autowired em todo o projeto, a qual realiza a injeção de dependência.

```
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.validation.Valid;
import java.security.Principal;
@Controller
public class ProdutoController {

    @Autowired
    private ProdutoService service;

    @Autowired
    private GrupoService grupoService;

    @Autowired
    private EstabelecimentoService estabService;

    @InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(Grupo.class, new GrupoEditor());
    }

    @RequestMapping("/secured/admin/produto/editar/{id}")
    public String editarProduto(@PathVariable("id") Long id, RedirectAttributes redirect,
                                Model model) {
        model.addAttribute("produto", service.findById(id));
        model.addAttribute("grupos", grupoService.findAll());
        return "produto/form";
    }

    @RequestMapping("/secured/admin/produto/novo")
    public String novoProduto(Model model) {
        model.addAttribute("produto", new Produto());
        model.addAttribute("grupos", grupoService.findAll());
        return "produto/form";
    }

    @RequestMapping("/secured/admin/produto/save")
    public String salvarProduto(@Valid Produto produto, BindingResult erros, Model model,
                                RedirectAttributes redirect) throws Exception {
        if (erros.hasErrors()) {
            model.addAttribute("grupos", grupoService.findAll());
            return "produto/form";
        }
        service.save(produto);
        return "redirect:/secured/produto/lista";
    }

    @RequestMapping("/secured/produto/lista")
    public String listarProdutosLojista(Model model, Principal user) {
```

```
        PessoaLojista lojista = (PessoaLojista) ((Authentication) user).getPrincipal();
        model.addAttribute("produtos", service.findByEstab(lojista.getEstab()));
        return "produto/lista";
    }

    @RequestMapping("/produto/lista")
    public String listarProdutosCliente(Model model, Principal user) {
        model.addAttribute("produtos", service.findAll());
        return "produto/lista";
    }

    @RequestMapping("/produto/estab/{id}/lista")
    public String listarProdutosClienteByEstab(Model model, @PathVariable("id") Long id)
    {
        Estabelecimento estab = estabService.findById(id);
        model.addAttribute("produtos", service.findByEstab(estab));
        return "produto/lista";
    }
}
```

Listagem 12 - Controller de produtos

Muitos métodos utilizam como parâmetro o “model”, pois possibilita o compartilhamento de informações entre o *controller* e as páginas HTML. O método “listarProdutosClienteByEstab” realiza uma busca no banco de dados por todos os produtos de determinado estabelecimento e disponibiliza essas informações à página HTML com o comando “model.addAttribute("produtos", service.findByEstab(estab));”, a qual fará a iteração da lista disponibilizada com o uso de JSTL e disponibilizará as informações em uma tabela, conforme exemplo da Listagem 13.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<!DOCTYPE html>
<html>
<head>
    <jsp:include page="../cabecalho.jsp"/>
    <title>Produtos</title>
</head>
<body>
<div id="wrapper">
    <jsp:include page="../menu.jsp"/>
    <div id="page-wrapper">
        <div class="row">
            <div class="col-lg-12">
                <h1 class="page-header">Produtos</h1>
```

```
    </div>
</div>
<div class="row">
    <div class="col-lg-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                Listagem de Produtos
            </div>
            <!-- /.panel-heading -->
            <div class="panel-body">
                <div class="dataTable_wrapper">
                    <table class="table table-striped table-bordered table-hover"
                           id="dataTables-example">
                        <thead>
                        <tr>
                            <sec:authorize access="isAnonymous()">
                                <th>Estabelecimento</th>
                            </sec:authorize>
                            <th>Descrição</th>
                            <th>Valor</th>
                            <th>Grupo</th>
                            <sec:authorize access="hasAnyRole('ROLE_ADMIN')">
                                <th>Editar</th>
                                <th>Excluir</th>
                            </sec:authorize>
                            <sec:authorize access="hasAnyRole('ROLE_USER')">
                                <th>Adicionar</th>
                            </sec:authorize>
                        </tr>
                        </thead>
                        <tbody>
                        <c:forEach items="${produtos}" var="produto">
                            <tr class="gradeU">
                                <sec:authorize access="isAnonymous()">
                                    <td>${produto.estab.nome}</td>
                                </sec:authorize>
                                <td>${produto.descricao}</td>
                                <td>${produto.valor}</td>
                                <td>${produto.grupo.descricao}</td>
                                <sec:authorize access="hasAnyRole('ROLE_ADMIN')">
                                    <td class="center">
                                        <a href="${pageContext.request.contextPath}
                                                    /secured/admin/produto/editar/${produto.id}">
                                            <button type="button" class="btn btn-primary btn-
                                                                                    xs">
                                                <i class="fa fa-arrow-circle-down">
                                                </i>
                                            </button>
                                        </a>
                                    </td>
                                    <td class="center">
                                        <a href="#">
                                            <button type="button" class="btn btn-primary btn-
                                                                                    xs">
                                                <i class="fa fa-trash">
                                                </i>
                                            </button>
                                        </a>
```

```
                                </td>
                            </sec:authorize>
                            <sec:authorize access="hasAnyRole('ROLE_USER')">
                                <td class="center">
                                    <a href="${pageContext.request.contextPath}
                                                /carrinho/adicionarItem/${produto.id}">
                                        <button type="button" class="btn btn-primary btn-
                                                                                    xs">
                                            <i class="fa fa-shopping-cart">
                                            </i>
                                        </button>
                                    </a>
                                </td>
                            </sec:authorize>
                        </tr>
                    </c:forEach>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>

<jsp:include page="../scripts.jsp"/>
</body>
</html>
```

Listagem 13 - Código HTML da página de listagem de produtos

O Spring MVC é um *framework action based* (baseado em ações) e neste projeto foi utilizado páginas JSP para o desenvolvimento da interface com o usuário. A integração do HTML (páginas) com o Boostrap é realizada principalmente por meio da utilização da propriedade “class” dos componentes, porém em alguns casos existe também a dependência de arquivos de Javascript, como no caso dos componentes de tabelas que possuem configurações como a de paginação, a busca e escolha de quantidade de registros mostrados.

4.4.6 Configuração e *deploy* do WildFly no IntelliJ

É necessário configurar o servidor de aplicações Wildfly na IDE IntelliJ para poder executar a aplicação e realizar o seu *deploy*. O primeiro passo é acessar o menu “Run > Edit Configurations” e clicar no botão “+” e localizar “JBoss Server > Local”, conforme demonstra a Figura 37.

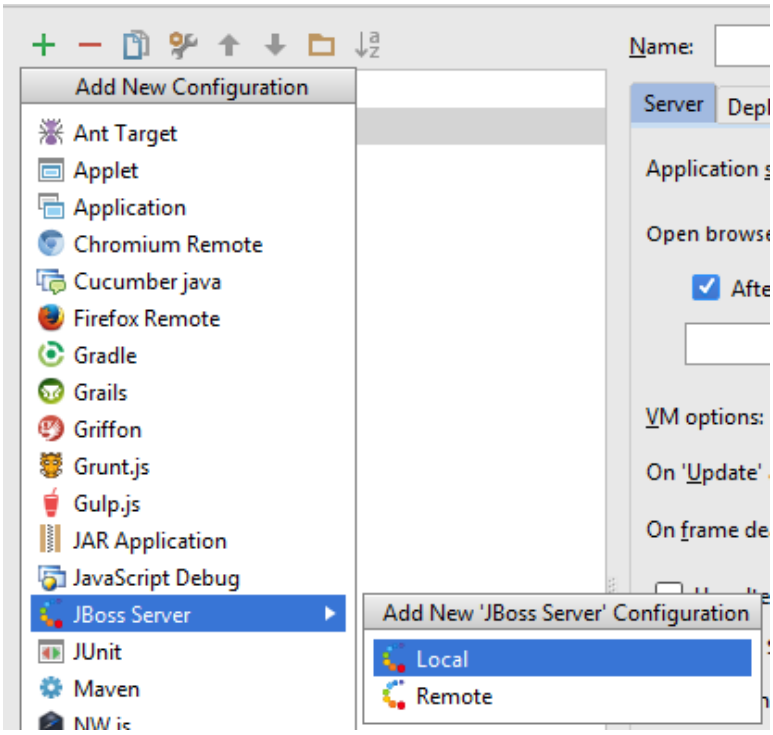

Figura 37 - Adicionar configuração JBoss no Wildlfy

Em seguida será necessário configurar o servidor conforme a Figura 38.

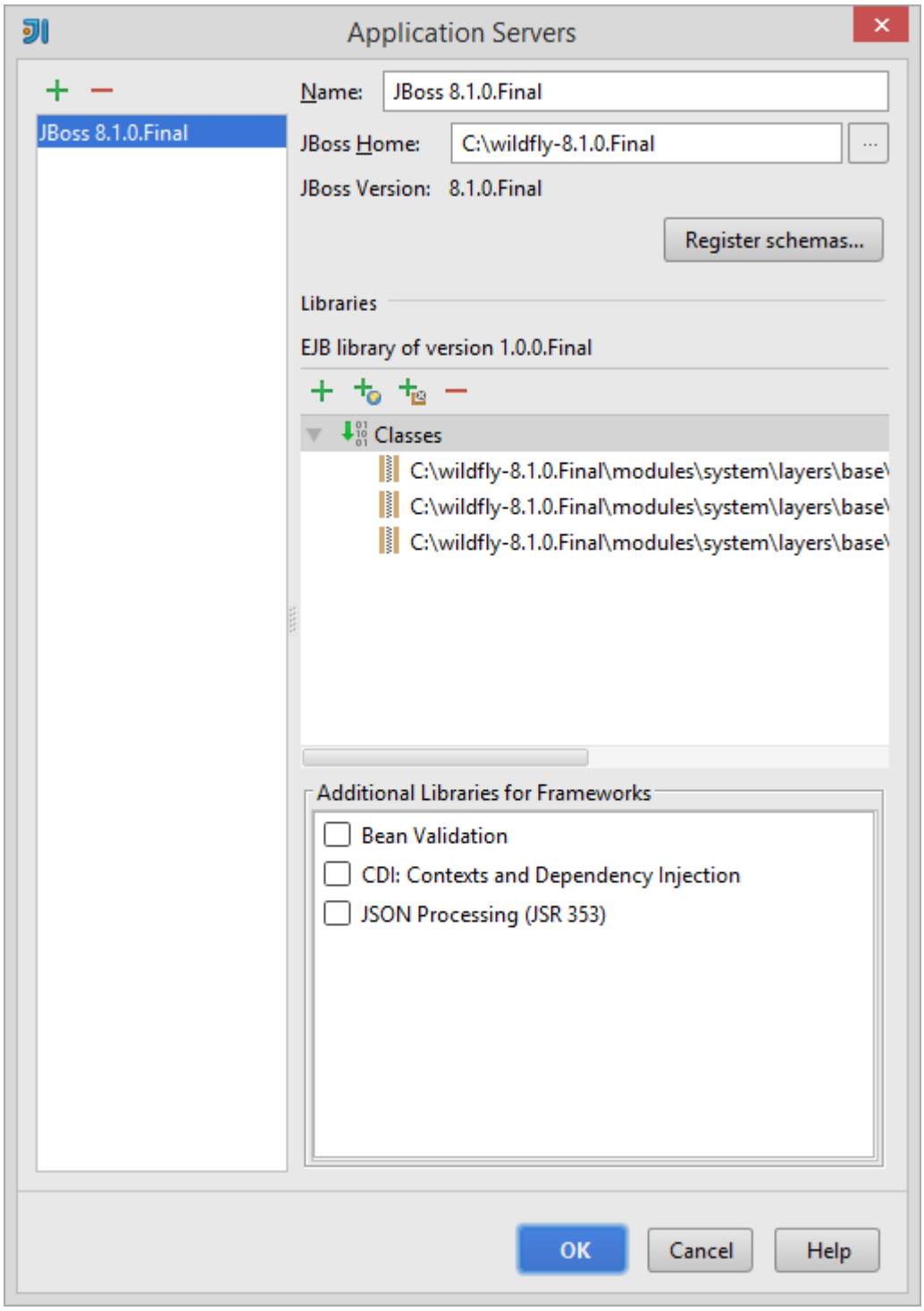

Figura 38 - Adicionando o Wildfly no IntelliJ

Continuando com as configurações, basta selecionar o servidor previamente cadastrado, adicionar o aplicativo para *deploy* na aba “Deployment” e alterar as configurações desejadas, conforme a Figura 39.

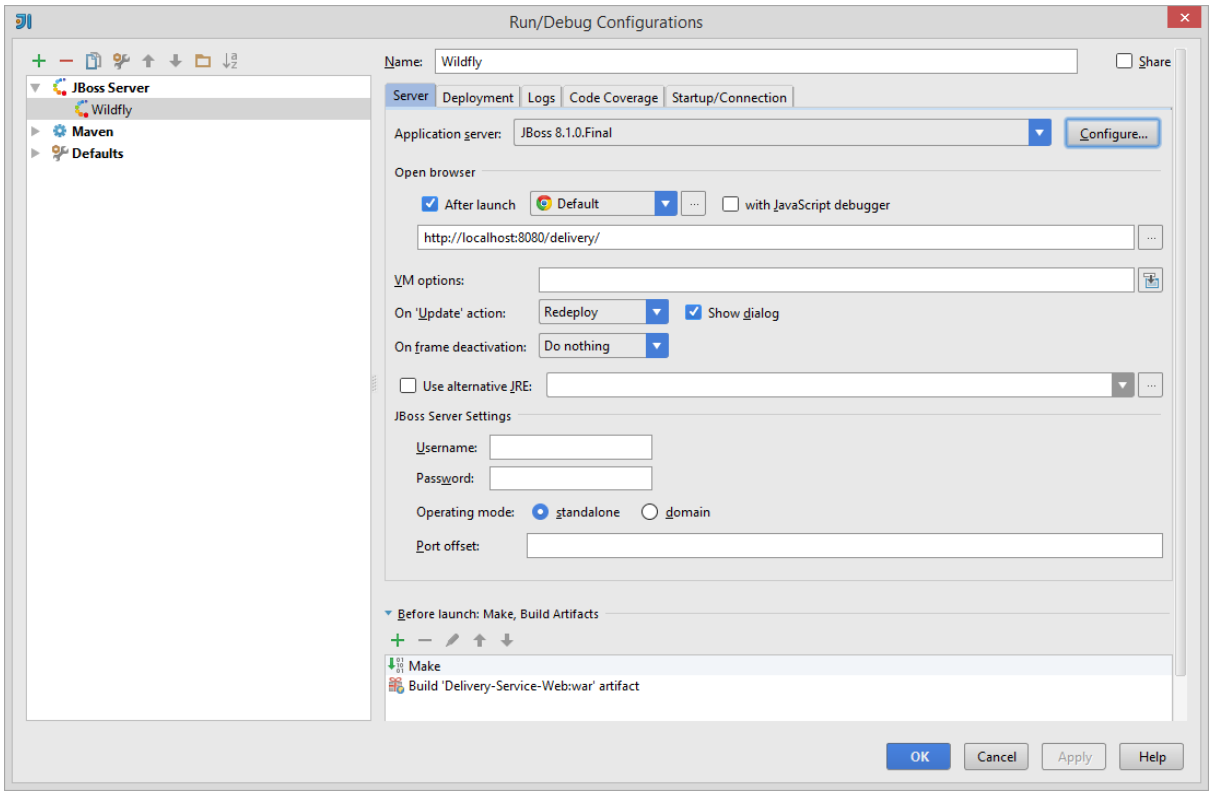

Figura 39 - Finalizando a configuração do Wildlfy no IntelliJ

O último passo é selecionar a configuração previamente cadastrada e realizar o *deploy* do projeto no IntelliJ selecionando o botão “Play” e acompanhando o log, conforme demonstra a Figura 40.

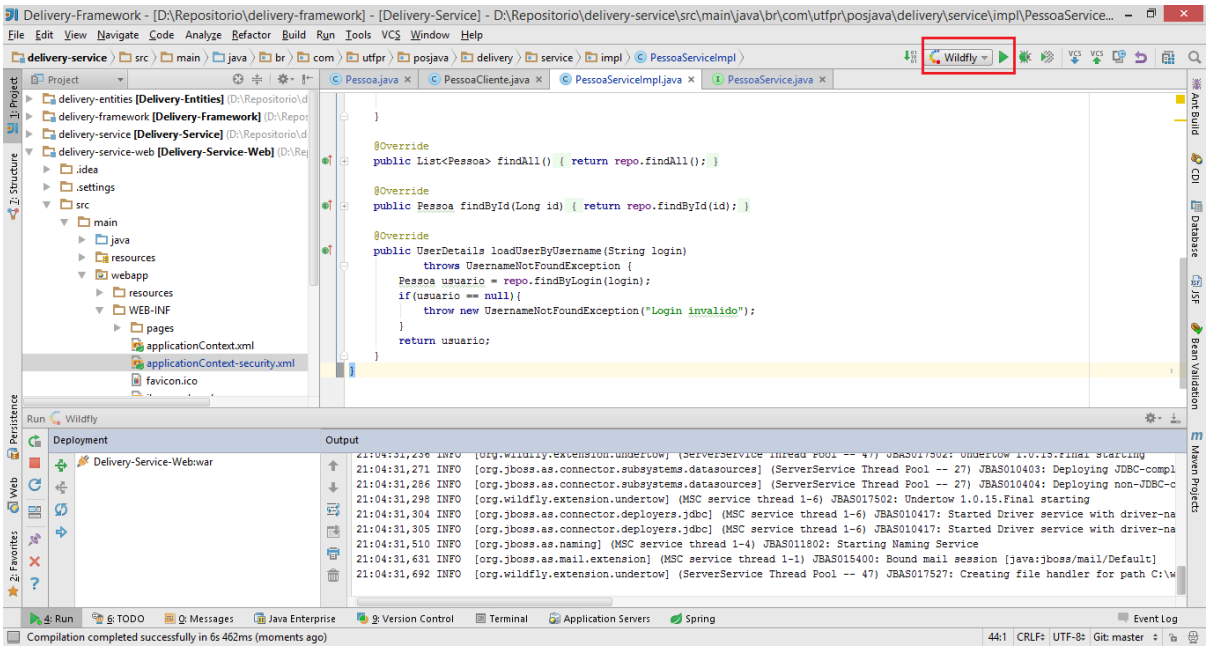

Figura 40 - *Deploy* da aplicação pelo IntelliJ IDEA

5 CONCLUSÃO

O objetivo deste trabalho foi apresentar um sistema *web* responsivo desenvolvido em linguagem Java. As maiores dificuldades enfrentadas ocorreram no início do projeto, no aspecto de configurações. Ao utilizar o IntelliJ IDEA, uma IDE completa e com muitos recursos, são necessárias várias configurações, algumas na IDE e outras no próprio projeto. Inicialmente foram criados os projetos utilizando a arquitetura do Apache Maven e configuradas as dependências, tanto de artefatos externos quanto entre os próprios projetos. Posteriormente realizou-se a configuração do servidor de aplicações WildFly combinado ao *deploy* do Apache Maven.

Assim procedeu-se para o *framework* Spring, tanto do Spring Framework quanto do Spring Security e Spring Web MVC. Apesar de ambos possuírem extensa documentação, o conjunto de tecnologias utilizadas no desenvolvimento, a exemplo do Apache Maven, dificulta a localização de informações específicas para serem aplicadas à estrutura de projeto utilizado. Após a realização dos testes de configurações de arquivos XML, bem como diversas trocas de URLs, as configurações atingiram o resultado esperado.

O aplicativo proposto condensa todos os esforços para a utilização e apresentação dos benefícios dos *frameworks* utilizados, tanto em *front-end* quanto em *back-end*. A agilidade proporcionada pelo *framework* Spring, tanto nos aspectos de classes de controle e padrões MVC quanto em consultas de banco de dados, minimiza drasticamente o tempo de desenvolvimento, permitindo que o desenvolvedor mantenha o foco nas regras de negócio. O HTML 5 e o Boostrap garantem a implementação de uma interface ao usuário de qualidade e dinâmica sem complicação.

Considerando o objetivo deste trabalho é possível afirmar que as tecnologias utilizadas possuem um grande impacto nos quesitos de qualidade e produtividade, visto que a implementação do sistema ocorreu em poucos meses e o resultado atingiu as expectativas e objetivos propostos.

REFERÊNCIAS

ADHIKARI, Ashok. **What is multi-tenant application?**. Disponível em http://whatwelearned2day.blogspot.com.br/2015/04/what-is-multi-tenant-application.html. Acesso em 25 mai. 2015.

ALTERMANN, Denis. **Design responsivo: entenda o que é a técnica e como ela funciona**. Disponível em: http://www.midiatismo.com.br/o-mobile/design-responsivo-entenda-o-que-e-a-tecnica-e-como-ela-funciona. Acesso em 24 mai. 2015.

BATTISTI, Julio. **Criando aplicações em 3, 4 ou n camadas.** Disponível em: http://www.juliobattisti.com.br/artigos/ti/ncamadas.asp. Acesso em 26 mai. 2015.

COSTA, Aécio. **Conceitos básicos sobre sistemas web: arquitetura e serviços**. Disponível em: http://www.aeciocosta.com.br/wp-content/uploads/FG/Projeto%20de%20Sistemas%20na%20Internet%202014-1/3-PSI-Conceitos%20e%20Arquiteturas%20da%20Web.pdf. Acesso em 24 mai. 2015.

PAULA FILHO, Wilson de Pádua. **Engenharia de software**. Rio de Janeiro: LTC, 2003.

GENTIL, Efraim. **Introdução ao Spring Framework**. Disponível em: http://www.devmedia.com.br/introducao-ao-spring-framework/26212 Acesso em 26 mai. 2015.

HIBERNATE. **Hibernate**. Disponível em: http://hibernate.org/. Acesso em 17 jun. 2015.

INTELLIJ. **IntelliJ IDEA**. Disponível em: https://www.jetbrains.com/idea/. Acesso em 02 jun. 2015.

IRISH, Paul. GARSIEL, Tali. **Como os navegadores funcionam: bastidores dos navegadores modernos**. Disponível em: http://www.html5rocks.com/pt/tutorials/internals/howbrowserswork/. Acesso em 28 mai. 2015.

JAVA. **O que é Java**. Disponível em: http://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/. Acesso em 02 jun. 2015.

KUROSE, James. F. & ROSS, Keith W. **Redes de computadores e a internet: uma abordagem top-down**, 3ª Edição, Editora Pearson, São Paulo– SP, 2006.

LOPES, Sergio. **Design responsivo por uma web única**. Disponível em: http://sergiolopes.org/responsive-web-design/. Acesso em 30 mai. 2015.

MAVEN. **Apache Maven project**. Disponível em: https://maven.apache.org/. Acesso em 17 jun. 2015.

MYSQL. **The world's most popular open source database**. Disponível em: https://www.mysql.com/. Acesso em 17 jun. 2015.

PEREIRA, Ana Paula. **O que é CSS?**. Disponível em: http://www.tecmundo.com.br/programacao/2705-o-que-e-css-.htm. Acesso em 24 mai. 2015.

OSORIO, Luis. **Aplicações web – conceitos**. Disponível em: http://pt.slideshare.net/lmcosorio/3-aplicacoes-web-conceitos. Acesso em 24 mai. 2015.

RADDAR. **Seu site já se adaptou à responsividade?**. Disponível em http://blog.raddar.com.br/ja-se-adaptou-a-responsividade/. Acesso em 20 mai. 2015.

SB ADMIN 2. **SB Admin 2**. Disponível em: http://startbootstrap.com/template-overviews/sb-admin-2/. Acesso em 17 jun. 2015.

SEO. **Web design responsivo**. Disponível em: https://developers.google.com/webmasters/mobile-sites/mobile-seo/configurations/responsive-design?hl=pt-br. Acesso em 01 jun. 2015.

SILVA, Maurício Samy. **Web design responsivo**. Disponível em http://www.livrosdomaujor.com.br/rwd/capitulo1.pdf. Acesso em 03 jul. 2015.

SILVEIRA, Guilherme. **Um produto para muitos clientes: implementando multitenancy**. Disponível em: http://blog.caelum.com.br/um-produto-para-muitos-clientes-implementando-multitenancy/. Acesso em 26 mai. 2015.

SIMONI, Rodrigo. **HTML 5 e o SEO: mitos e verdades**. Disponível em: http://tableless.com.br/html-5-e-seo-mitos-e-verdades/. Acesso em 29 mai. 2015.

SPRING DATA JPA. **Spring Data JPA**. Disponível em: http://projects.spring.io/spring-data-jpa/. Acesso em 02 jun. 2015.

SPRING FRAMEWORK. **Spring Framework**. Disponível em: http://projects.spring.io/spring-framework/. Acesso em 02 jun. 2015.

SPRING SECURITY. **Spring Security**. Disponível em: http://projects.spring.io/spring-security/. Acesso em 02 jun. 2015.

TAURION, Cesar. **Entendendo o modelo multitenancy**. Disponível em: http://imasters.com.br/artigo/19067/cloud/entendendo-o-modelo-multi-tenancy/. Acesso em 25 mai. 2015.

TERUEL, Evandro Carlos. **HTML 5: Guia Prático**, 1ª edição, São Paulo: Érica, 2011.

WILDFLY. **WildFly**. Disponível em: http://wildfly.org/. Acesso em 17 jun. 2015.

WORKBENCH. **MySQL Workbench**. Disponível em: https://www.mysql.com/products/workbench/. Acesso em 02 jun. 2015.

W3C. **Consórcio World Wide Web**. Disponível em http://www.w3c.br/Home/WebHome. Acesso em 05 ago. 2015.