UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMA DE INFORMAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO DESENVOLVIMENTO DE UMA ARQUITETURA ESCALÁVEL PARA A CONSTRUÇÃO DE SISTEMAS WEB UTILIZANDO JAVA

ROMILDO JOZUÉ PAITER

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMA DE INFORMAÇÃO

RELÁTORIO DE ESTÁGIO SUPERVISIONADO DESENVOLVIMENTO DE UMA ARQUITETURA ESCALÁVEL PARA A CONSTRUÇÃO DE SISTEMAS WEB UTILIZANDO VRAPTOR

ROMILDO JOZUÉ PAITER

Relatório apresentado ao Instituto de Computação da Universidade Federal de Mato Grosso, para obtenção do título de Bacharel em Sistema de Informação.

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMA DE INFORMAÇÃO

ROMILDO JOZUÉ PAITER

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Sistema de Informação como uma das exigências para obtenção do título de Bacharel em Sistema de Informação da Universidade Federal de Mato Grosso

Aprovado por:

Prof. MSc. Nilton Hideki Takagi
Instituto de Computação
(Orientador e Coordenador de Estágio)

Esp. William Chitto de Souza Pinto
RCF Inovações
(Supervisor)

Prof. MSc. Raphael de Souza Rosa Gomes
Instituto de Computação
(Professor convidado)

DEDICATÓRIA

*A Deus graças pelas forças que foram dados e as bençõe*s que me ocorreram durante todo o trajeto que foi percorrido.

À todos aqueles participaram diretamente e indiretamente para a conclusão desse curso.

AGRADECIMENTOS

Primeiramente a Deus, pela vida, saúde e inspiração que me permitiram escrever este trabalho.

Aos meus pais, Romualdo e Marilze Paiter, que sempre me incentivaram a estudar e lutar pelos meus objetivos.

A minha esposa, Camila, por sempre me apoiar em tudo o que eu faço e por compreender o tempo em que estive ocupado me dedicando a esta graduação, que passou por diversas etapas. Aos meus filhos Guilherme e Miguel Paiter, que me deram forças para continuar.

Aos grandes amigos Leandro Ferreira e Willian Chitto que me passaram grandes conhecimentos enquanto estava-mos trabalhando juntos e mesmo após com nossas conversas.

Também aos companheiros e amigos de curso Lucas Abido e Jairo Magesti, onde formamos uma equipe com o mesmo objetivo, se formar.

Aos professores, que transmitiram seu conhecimento, experiência com paciência conosco, permitindo que nos tornássemos profissionais dessa área tão competitiva e acirrada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
CPU	*Central Processing Unit*
UML	*Unified Modeling Language*
ORM	*Object-relational mapping*
ASI	Arquitetura de Sistema da Informação
SI	Sistema de Informação
DI	*Dependency Injection*
IDE	*Integrated Development Environment*
JSP	*Java Server Pages*
IoC	*Inversion of Control*
EJB	*Enterprise JavaBeans*
IoD	*Injection of Dependency*
ID	Injeção de Dependência
SGDB	Sistema de gerenciamento de banco de dados
SaaS	*Software as a service*

RESUMO

O presente documento retrata o levantamento realizado para a construção de um sistema voltado a comercialização de veículos na internet, possivelmente denominado de portal automotivo. Este sistema possui as funcionalidades de cadastro de usuário dos recursos, cadastro de anúncio, cadastro completo do perfil do usuário, exibir o anúncio e busca de anúncios pelo internauta seja ele pessoa física ou jurídica. Para a realização desse trabalho foram utilizados os seguintes componentes do sistema, linguagem Java, banco de dados PostgreSQL, *Framework* MVC Vraptor 4, injeção de dependência com CDI implementado pelo Weld 1.1, controle de persistência utilizando JPA com Hibernate 4.3.0, controle de dependências do projeto com Apache Maven e utilizado como IDE o Eclipse para a construção do projeto.
Foram criados diagrama de classe e de uso, utilizando modelagem UML com a ferramenta Astah Professional. Também será apresentado os resultados obtidos e dificuldades encontradas durante o tempo da realização do estágio supervisionado.

Palavras Chaves:
Java, Framework VRaptor, Hibernate, Postgres, CDI, JPA, Weld

INTRODUÇÃO

No contexto atual da tecnologia, o desenvolvimento de Sistemas de Informação (SI) tem recebido diversas demandas para solucionar problemas existentes. A partir dessa necessidade, o objetivo desse trabalho é estruturar e documentar o processo a construção de uma arquitetura de software que utilize padrões de projeto e princípios básicos da orientação a objeto.

A arquitetura de uma aplicação acontece por decisões que tomamos para a construção das atividades do sistema, outra forma de pensar sobre a arquitetura de maneira simples, é, se ao alterar algum código e "ver" muita complexidade para se efetuar a mudança, logo, isso faz parte da arquitetura do projeto. Portanto a arquitetura se resume em artefatos importantes do projeto (FOWLER, 2006).

Com este trabalho foi adquirido uma consolidação de todo o conteúdo visto no curso de Sistema de Informação e nas leituras para a construção de um sistema sólido.

Todo sistema solicitado tem um objetivo diferente uns dos outros, mas em todos, os requisitos arquiteturais são muito parecidos, como exemplo, o armazenamento de informações atualmente também chamada de persistência, o controle de acesso de usuários, autenticação de usuários, fornecimento de relatórios e interface humano computador. A mudança entre cada sistema se dá geralmente através de regras de negócios e domínios individuais de cada solução.

Não há maneira certa ou errada de projetar um software. Você aprende como projetar construindo exemplos de projetos existentes e discutindo seu projeto com outros (SOMMERVILLE, 2011).

Através dos diversos padrões de software apresentados pelos mais diferentes autores, percebe-se que tratam de soluções de problemas recorrentes no desenvolvimento de sistemas.

Com essa visão demonstrarei a construção de um projeto de software para criar uma arquitetura de um portal automotivo, onde deverão serão

mantidos usuários, lojas, anúncios de veículos, serviços e solicitação de pedido de atendimento.

Este trabalho apresenta um estudo sobre as soluções adotadas para o desenvolvimento de uma aplicação voltada a comercialização de veículos. O principal objetivo dessa aplicação é se tornar uma ferramenta de comercialização de produtos na área automotiva, como veículos, serviços, peças e acessórios inicialmente abrangendo a capital de Mato Grosso e municípios próximos.

Para atingir esse objetivo, o sistema entrará em produção de forma BETA, onde fornecerá algumas funcionalidades mas não todas logo em seu início tendo somente os serviços mais importantes disponibilizados. Forneceremos alguns serviços gratuitamente como estratégia para a captação dos primeiros usuários.

O *template/layout* utilizado foi adquirido por um site especializado, o Theme Forest, onde são vendidos *templates* de todas as áreas, bastando apenas escolher qual plataforma você usará ou se utiliza somente o formato em HTML.

A partir desse ponto vamos apresentar os estudos e resultados obtidos com a criação dessa nova ferramenta.

1. A ARQUITETURA DO PROJETO

Segundo a especificação da UML, citada por Flávia Delicato (2008, BASS et.al. 1996), pode ser concluída que a arquitetura de software é a estrutura organizacional do software. Uma arquitetura pode ser recursivamente decomposta em partes que interagem através de interfaces.

A arquitetura define como são organizados os elementos para montar o software e o arquiteto é o responsável pela sua criação. Mas o mais importante é saber quais as possíveis escolhas e que fatores influenciam na sua seleção (XAVIER, 2013).

A arquitetura criada não é somente para a discussão, mas para um ambiente corporativo, as informações terão a necessidade de serem armazenadas ou persistidas em um banco de dados. Segundo Fowler (2006) os dados são persistentes porque precisam estar disponíveis entre múltiplas execuções do programa e esses dados dever existir por vários anos.

Decisões de desenvolvimento de uma aplicação, envolvem vantagens e desvantagens, que o desenvolvedor deve conhecer ao utilizar padrões e *frameworks* em seus projetos. O desenvolvedor que desconhecer o lado negativo de determinada abordagem, sofrerá as consequências durante a manutenção quando surgirem na sua aplicação. Contudo o sucesso da arquitetura de um sistema está fortemente ligado com o design e a implementação (SILVEIRA, 2011).

E afinal, o que é arquitetura de software? É o termo utilizado para o conhecimento dos principais elementos de um sistema, tudo aquilo que é difícil de alterar, pode se dito que faz parte da fundação da arquitetura do sistema (FOWLER, 2006).

1.1. Projeto de Arquitetura

Projeto de arquitetura é um processo criativo onde é projetado a organização de um sistema de forma a satisfazer os requisitos funcionais e não funcionais de um sistema. Durante o processo de se projetar de arquitetura, os arquitetos de sistemas precisam tomar uma série de decisões estruturais, que afetam profundamente o sistema e seu processo de desenvolvimento (SOMMERVILLE, 2011).

Outra parte do projeto de arquitetura é levantado por Dennis e Wixon (2011) que não basta ter um projeto de arquitetura lógico, mas também é necessário ter o projeto de arquitetura físico que descreve o hardware, o software e o ambiente de rede do sistema.

Todo projeto de software independente do tamanho, merece ser mensurado para se ter informações do impacto na utilização de cada recurso. Isso vale também para a utilização dos padrões de arquitetura no projeto. Cada padrão de projeto tem a sua especificação de uso que deve ser analisada e conhecida suas regras.

1.2. Sobre padrões de projetos

Segundo a equipe K19 Treinamentos (2012), os padrões de projeto são soluções consolidadas dentro do mundo corporativo de produção de software que resolvem problemas recorrentes na construção e manutenção de softwares.

Um padrão de projeto descreve um problema que é encontrado em nosso ambiente de trabalho, também apresenta uma solução para esse problema, solução essa que pode ser utilizada quantas vezes forem necessárias sem precisar ser alterada (ALEXANDER, 1977).

Com a utilização de, Padrões de Projeto ou *Design Pattern* todos software devem manter um padrão. Esse padrão facilita a manutenção e ajuda na evolução do software.

1.3. Padrões Arquiteturais

Segundo Sommerville (2011), padrões arquiteturais são como uma descrição abstrata e estilizada, de boas práticas experimentadas e testadas em diferentes sistemas e ambientes. Dessa forma o padrão de arquitetura deve descrever uma organização de sistema bem-sucedida em sistemas anteriores.

Diz ainda que os padrões de arquitetura foram propostos na década de 1990 com o nome de 'estilos de arquiteturas' (SHAW e GARLAN, 1996).

1.3.1. Arquitetura de repositório

O padrão Repositório pode ser descrito com um conjunto de componentes que interagem para compartilhar dados.

Nesse modelo podemos citar diversas ferramentas, um exemplo comum é o uso de uma IDE que controlará diversos tipos de arquivos e manterá a versão desse arquivos apoiando um desenvolvimento voltado a modelos. Esse repositório no caso pode ser um ambiente controlado de versões, que mantém o controle das alterações de software e permite a reversão das versões anteriores. (Sommerville, 2011)

A Figura 1, demonstra o funcionamento de um repositório, com diversos tipos de arquivos centralizados em um único lugar.

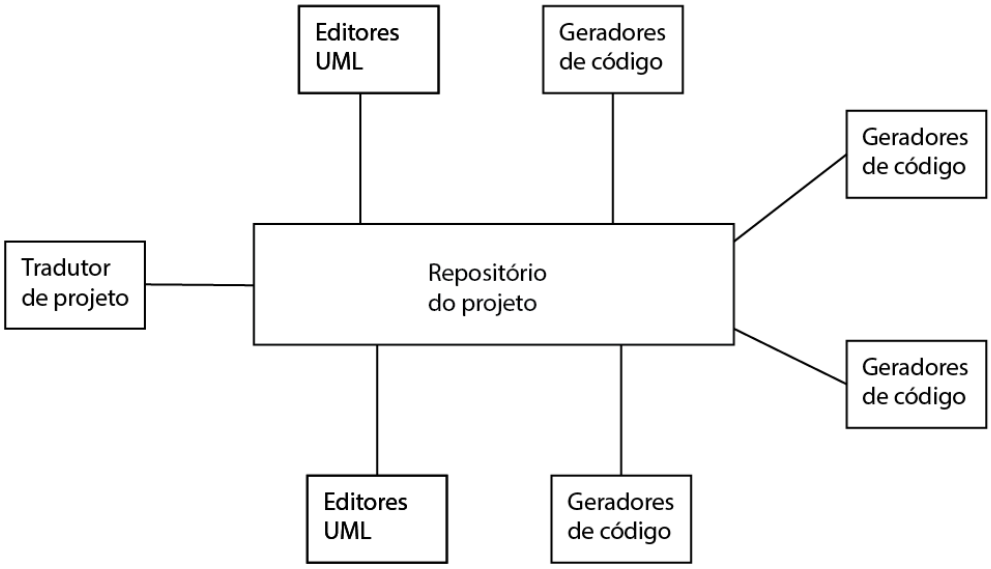

Figura 1: Exemplo de uma arquitetura de repositório para um IDE

1.3.2. Arquitetura de cliente-servidor

A arquitetura cliente-servidor é um modelo de sistema organizado em serviços disponibilizados em servidores e acessados clientes associados que utilizam esses serviços (SOMMERVILLE, 2011).

Como componentes que fazem parte desta arquitetura citamos:

1. Um apanhado de servidores que oferecem serviços para outros subsistemas. Como exemplo, servidores de impressão, servidores de arquivos e servidores de compiladores.

2. Um conjunto de clientes que utilizam os serviços fornecidos pelos servidores. Atualmente pode-se ter diversos tipos de clientes, computadores, smartphones e outros. Além de poder existir diversas instâncias rodando ao mesmo tempo.

3. É necessária uma rede que forneça a esses clientes acesso aos dados.

1.3.3. Arquitetura em camadas

Para Fowler (2006), a arquitetura em camadas é uma técnica amplamente utilizada em projetos, pois é uma das maneiras mais fáceis de particionar um sistema complexo. Ao se trabalhar em camadas logo pensa-se em subsistemas que compõem o sistema principal. A demonstração direta do uso de camadas, é onde a camada superior utilizará a camada mais inferior, evitando acessar diretamente uma camada mais profunda.

A divisão em camadas traz diversos benefícios:

1. Compreensão da camada como um todo sem saber exatamente o que faz a camada inferior.
2. Pode ser alterada a tecnologia de cada camada com mais facilidade.
3. É mais fácil minimizar as dependências entre as camadas.
4. Utilização de camadas é excelente para padronizações.
5. Após a construção de uma camada ela pode ser utilizada por diversos níveis.

E como aspecto negativo da utilização de camadas, tem-se:

1. As vezes quando uma alteração for necessária pode-ra se tornar um processo em cascata.
2. A utilização de uma camada extra que represente pouco pode prejudicar o desempenho.

1.4. Objetivos de uma arquitetura de software

O objetivo da utilização da arquitetura de software, segundo Jonhson (2002) é tornar o software robusto, performático, escalável e aproveitando princípios de *design* da orientação a objetos, diminuir a complexidade possibilitando a fácil manutenção e extensão, além da pontualidade na entrega e promover a reutilização de código.

2. TECNOLOGIA E TÉCNICAS UTILIZADA

2.1. Plataforma JAVA

O desenvolvimento de uma aplicação sob plataforma JAVA fornece benefícios apresentados por Silveira (2011) onde o uso da plataforma engloba portabilidade e segurança além fornecerem investimentos das grandes empresas de SI no mundo. Tem alta aceitação como plataforma de grandes bancos e empresas que necessitam ter maior facilidade de mudança. Esta facilidade evita que a tecnologia fique como o efeito *Vendor Lock-in* (Único Fornecedor), permitindo que não se fique dependentes de um único fabricante, um único sistema operacional ou um único banco de dados.

Atualmente a plataforma Java é a mais utilizada no mundo, foi realizado um estudo ponderando 12 questões em 10 fontes de pesquisa, analisando os seguintes termos: interesse, demandas por emprego, popularidade em fóruns, redes sociais, pode assumir o controle da linguagem e criação de seu próprio projeto. Java obteve 100 pontos na pesquisa, o que a deixou em primeiro lugar como linguagem que trabalha para web, mobile, empresarial e embarcado (Cass et. d, 2014).

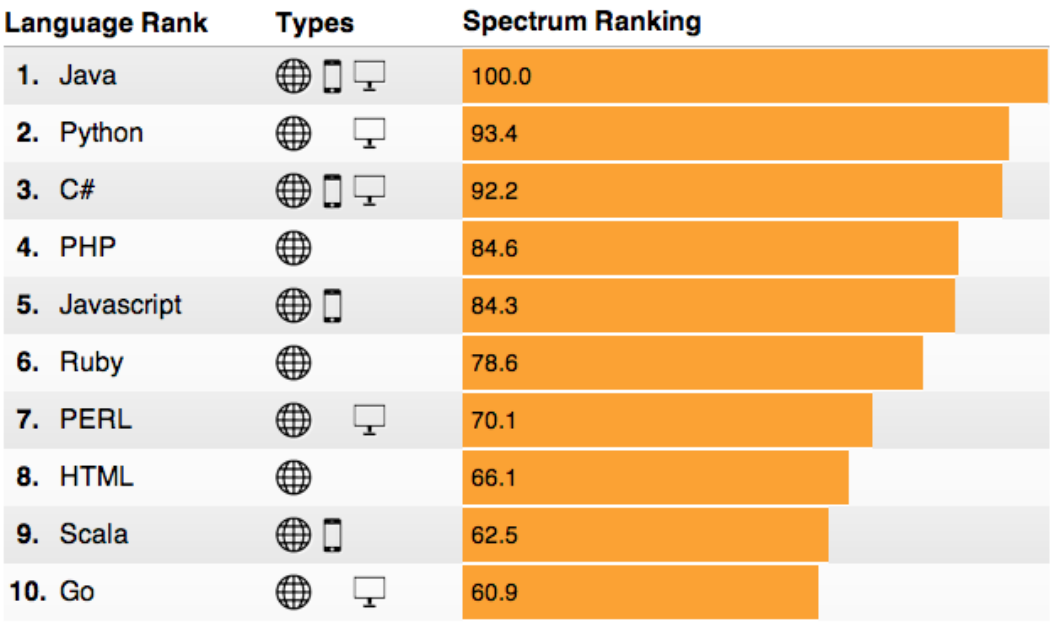

Figura 2: Ranking em pontos das linguagens mais relevantes somente selecionado para web.

2.2. MVC

Model-View-Controller (MVC) é um conceito de engenharia de software para desenvolvimento e *design*, que propõe a separação de uma aplicação em três partes distintas: modelo, visão e controle. O modelo é usado para definir e gerenciar o domínio da informação, a visão é responsável por exibir os dados ou informações da aplicação e o controle controla as duas camadas anteriores, exibindo a interface correta ou executando algum trabalho complementar para a aplicação (GONÇALVES, 2007).

A camada de apresentação é uma escolha independente das demais camadas abaixo ficando o desenvolvedor responsável em escolher entrar uma interface rica ou uma interface por navegador web (FOWLER, 2006).

O MVC é o padrão de arquitetura de software mais recomendado para aplicações interativas (SINGH, 2002).

2.3. Vraptor 4

É um *framework* brasileiro, criado pelos irmãos Paulo Silveira e Guilherme Silveira, em 2004 na Universidade de São Paulo. Em 2006 foi lançada a sua primeira versão estável, o Vraptor 2 com ajuda de Fabio Kung,

Nico Steppat e outros desenvolvedores, que absorveram ideias e boas práticas do Ruby on Rails (SILVEIRA, 2011).

O objetivo principal da construção desse framework foi fugir de *frameworks* complexos da época e que demandavam escrever muitos arquivos de configuração, facilitando assim o desenvolvimento web sem se tornar refém de suas classes e interfaces. (CAVALCANTI, 2014)

Atualmente o Vraptor se encontra na versão 4, conta com o suporte às diversas tecnologias de ponta da linguagem Java. Este framework possibilita a criação e serviços, utilização de CDI e ainda possui plugins desenvolvidos pela comunidade que facilitam a vida do desenvolvedor.

Com uma comunidade grande, atualmente vem ganhando cada vez mais mercado. É um *framework* que tem entrado no mundo corporativo para o desenvolvimento de novas aplicações (VRAPTOR, 2014).

Os principais motivos para a sua aceitação e utilização são: Alta produtividade, Testabilidade, SOA e Rest prontos e fáceis de implementar, CDI, Documentação em português, Economia, Melhores prática de desenvolvimento e Curva de aprendizado muito rápida (VRAPTOR, 2014).

Um dos diferenciais é que se pode trabalhar melhor com a orientação a objetos, pois não cria restrições para o design das classes. A camada de visualização, também objeto de estudo, pois temos total liberdade para escolher qual tecnologia utilizar nesta camada, podendo usar JSP, Velocity, Freemarker que gerenciam os templates. Os componentes visuais devem ser feitos manualmente, ou ainda usando bibliotecas externas como o jQuery UI, Angular JS, Bootstrap ou ExtJS (CAVALCANTI, 2014).

2.4. Pré-requisitos

O Vraptor tem como pré-requisitos principais o Java 7 e o CDI 1.1 ou superiores. Para se rodar a aplicação devemos utilizar servidores com suporte a estas especificações. Atualmente os servidores que suportaram estas configurações são o: Glassfish 4, Wildfly 8, Tomcat 7/8 e Jetty 8, sendo que é necessário a configuração o Weld 2.0 nos dois últimos servidores e nos outros servidores ele já vem diretamente configurado em sua arquitetura.

A Figura 3, demonstra as linhas de configuração que devemos adicionar ao arquivo WEB-INF/web.xml para efetuar a configuração do CDI no Tomcat, utilizando a implementação do Weld 2.0.

```
1 <listener>
2       <listener-class>[
3         org.jboss.weld.environment.servlet.Lis-
tener
4       </listener-class>
5 </listener>
```

Figura 3: Exemplo de configuração do listener do Weld 2.0 no Tomcat.

2.5. Inversão de Controle

Para Johnson (2005) a Inversão de Controle (Inversion of Control – IoC) IC, esta diretamente associada ao"Principio de Hollywood" *("Hollywood Principle")* que basicamente é "Não me chame, Eu chamarei você" *("Don`t call me, I`ll call you")*, este conceito aborda diversas coisas, EJB, Servlet e qualquer classe que possa ter um retorno. Ainda comenta que é a utilização de IC é de fundamental importância para os *framewoks* pois diminui o acoplamento entre as classes, resolvendo as dependências com configurações no s métodos e construtores em tempo de execução.

MACHADO (2008), especifica que o IC trabalha diretamente na mudança do fluxo de controle de um programa, onde o desenvolvedor determina quem deve ser executado para determinado evento e não a passagem do procedimento diretamente.

2.6. Injeção de Dependência

A Injeção de Dependência ("Injection of Dependency" IoD) ID começou no Java EE 5, como um mecanismo para fornecer um componente de instanciação de forma automática para os objetos.

A ID se tornou comum na programação, onde um agente externo, que pode ser chamado do *conteiner* ou servidor web, é responsável por gerenciar e solucionar as relações entre os componentes e suas dependências, com o EJB

3.0. Os componentes de referência de uma dependência são utilizados por meio de uma anotação, quem solucionará essa anotação será o *conteiner* e assim garantir em tempo de execução que o objeto associado será referenciado ao objeto correto. Isso é normalmente feito por uma implementação de reflexão. (COULOURIS, 2013)

2.7. CDI

Segundo Cordeiro (2013), CDI é uma especificação Java, que o termo original é *(Context and Dependency Injection for Java EE)*. É um padrão de configuração simples, utilizando anotações, que não analisa os tipos de objetos.

A criação de especificações define como deve ser a implementação para o uso de uma tecnologia, fazendo assim que empresas, fundações, grupos de usuários e pessoas tenham a liberdade de implementar da maneira que desejam, mas desde que seja seguida a especificação. Para garantir que o funcionamento da implementação esteja correta existe uma bateria de testes que são aplicados a uma implementação com o intuito de certificar que foi realizada corretamente.

2.7.1 Configurando o CDI

A forma de configuração adotada no projeto foi a utilização de uma ferramenta de gerenciamento de dependências no projeto, o Apache Maven. Esta ferramenta de gerenciamento de projetos de software é baseada no conceito de um modelo de objeto do projeto (POM), que irá controlar a construção do projeto, e a elaboração de relatórios e documentação de uma peça central de informações. (MAVEN, 2014)

Para configurar o CDI em seu projeto são necessários os seguintes passos:

1. Adicione o trecho da Figura 4 dentro da marcação *dependencies*, no arquivo *pom.xml* para que faça o download do arquivo cdi-api.jar. O mesmo contem a especificação do CDI, com isso será incluído na aplicação e utilizado no contexto do servidor.

```
<dependency>
    <groupId>javax.enterprise</groupId>
    <artifactId>cdi-api</artifactId>
    <version>1.1</version>
</dependency>
```

Figura 4: Adicionando a dependência do CDI no projeto via Maven

2. Crie um arquivo *beans.xml* na pasta META-INF dentro da pasta *resources* do projeto. A Figura 5 demonstra como fica o caminho do arquivo.

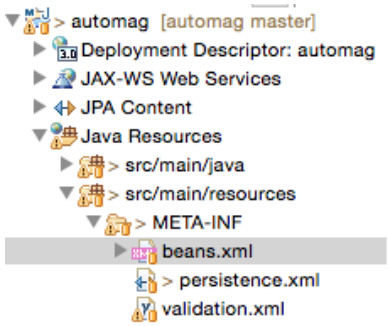

Figura 5: Exemplo da localização do arquivo beans.xml dentro de um projeto que utiliza maven.

3. A Figura 6 demonstra o conteúdo do arquivo *beans.xml*. A inclusão desse aquivo já habilitará o uso do CDI no seu projeto. Só o fato de existir esse arquivo já habilita o servidor a escanear as classes automaticamente. (LOPES, 2012)

Observação: O arquivo *beans.xml* também pode ser totalmente vazio.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                            http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
        version="1.1" bean-discovery-mode="all">

</beans>
```

Figura 6: Modelo de configuração do beans.xml para ativar o CDI no projeto.

O uso de CDI com anotações, a Figura 7 traz um exemplo da injeção de alguns objetos na classe `UsuarioController.java`, que está anotada com *@Controller*, que é um *stereotype* (estereótipo) é um tipo de definidor de comportamento do CDI.

```java
@Controller
public class UsuarioController {

    private UsuarioBusiness useBusiness;
    private Result result;
    private UsuarioValidator validator;

    @Deprecated
    public UsuarioController() {
    }

    @Inject
    public UsuarioController(UsuarioBusiness useBusiness, Result result, UsuarioValidator validator) {
        this.useBusiness = useBusiness;
        this.result = result;
        this.validator = validator;
    }
```

Figura 7: Classe UsuarioController do sistema Automagz, demonstrando a anotação @Inject que faz a injeção de dependências das Classes UsuarioBusiness, Result e UsuarioValidator diretamente no construtor.

2.8. JPA

JPA *(Java Persistence API)* é a especificação que define como efetuar a persistência dos dados em um banco de dados (KING, 2007).

Persistência é o fato de se gravar as informações para se recuperar posteriormente estas informações e assim manter um histórico das informações que o sistema trata.

Existem diversas implementações da API de JPA como, Hibernate, Eclipse Link, TopLink e outras nem tão conhecidas com o Mybatis que oferecem suporte ao ORM *(Object-relational mapping)*, sendo traduzido para Mapeamento Objeto/Relacional.

ORM é a persistência automatizada dos objetos em uma aplicação Java para as tabelas de um banco de dados relacional, fazendo uso de metadados que descrevem o mapeamento entre os objetos e o banco de dados. Seu objetivo é trabalhar com a transformação dos dados nas duas vias, tanto gravando informações nas tabelas, quanto recuperando essas informações para um objeto referente. (KING, 2007)

2.9. Hibernate

Segundo King (2007), criador do Hibernate, é uma ferramenta de mapeamento objeto/relacional completa que oferece todos os benefícios de um ORM.

O Hibernate da liberdade aos desenvolvedores para escreverem aplicações mais facilmente, diminuindo a preocupação com a persistência dos dados. Por ser um *"JPA Provider",*tem a implementação da especificação do JPA dando suporte para aplicações Java SE e Java EE.

Como benefícios do uso do Hibernate em projetos:

1. Alta Performance. O Hibernate habilita o uso de consultas atrasadas, diversas estratégias de recuperação dos dados, efetua o versionamento da informação.

2. Escalável: Foi desenvolvido para trabalhar com aplicações em servidores de cluster e entregar uma arquitetura escalável.

3. Confiabilidade: É muito conhecido por sua excelente estabilidade e qualidade, comprovando pela aceitação em uso de diversos projetos.

4. Extensibilidade: É altamente configurável e extensível.

Para habilitar o uso do Hibernate em um projeto que usará JPA é necessário adicionar ao nosso projeto as seguintes dependências Hibernate-Core, Hibernate-EntityManager apresentadas na Figura 8. O Hibernate-Validator-Cdi é uma extensão portátil para fornecer validações das classes do projeto ou até mesmos criar validadores personalizados. (HIBERNATE, 2014)

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.0.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.0.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator-cdi</artifactId>
    <version>5.1.0.Final</version>
</dependency>
```

Figura 8: Dependências necessárias para habilitar o uso de Hibernate no projeto.

2.10. Escalabilidade

Um sistema escalável quer dizer que o software tem grande potencial para crescimento, para assim ser considerado escalável. (CARVALHO, 2014)

Isso torna o sistema mais maleável ao crescimento da aplicação. Vale explicar que escalabilidade não é performance, escalabilidade é atender qualquer quantidade de acessos com uma mesma performance. Para melhorar a performance existem outras abordagens, como o uso de cache.

Algumas das funcionalidades que podemos aplicar a uma arquitetura escalável e que ajudarão a manter os serviços prestados com mais disponibilidade são:

1. *Stateless*: Um sistema onde a requisição não fique presa a um nó cliente, fazendo com que o tempo de resposta seja a menor possível.
2. *Request Response Time (RRT)*: Na construção de uma página podemos ter diversos componentes sendo eles js, CSS, imagens, onde os quais podem aumentar o tempo de resposta da página, portanto o RRT tem que ser trabalhado para que possa ser entregar todo o conteúdo da página com o menor número de bytes possíveis.
3. Cache: Esta é uma excelente maneira de melhorar o RRT, fazendo com que armazene as páginas que tenham muito acesso e assim diminuindo as requisições ao banco de dados e só alterando essa página quando ocorrer alguma alteração em suas informações.
4. *Remote Date:* É manter as informações separadas, o servidor de banco de dados estar separado do servidor de aplicação. Podemos utilizar serviços de SaaS (*Software as a service*) como o Amazon S3.
5. *Proxy* Reverso: Utilizado para mascarar as técnicas de caching e principalmente utilizado para esconder todos os servidores que possam estar envolvidos no *Front-end* da aplicação. Mas também oferece riscos ao adicionar esse serviço SPoF (Single Point Of Failure, Ponto simples de falha).

O uso desses recurso não quer dizer que é a resposta final, mas é uma maneira de dar o mínimo da escalabilidade ao sistema. (URUBATAN, 2014)

2.11. Hospedagem e Publicação das imagens

A hospedagem é um serviço publicação na internet. Para realizar esse serviço foi feito o contrato da empresa **Digital Ocean**.

E para aumentar a performance do serviço fornecido foi optado a utilização de um serviço externo para publicação de imagens.

O serviço escolhido é fornecido pela empresa **Cloudinary**, esta empresa fornece uma API completa para se fazer o envio das imagens para o seu servidor.

3. O PROJETO E A METODOLOGIA DE DESENVOLVIMENTO

Nesta sessão será apresentado mais sobre o projeto criado, como foi formado a sua arquitetura, quais foram as tecnologias utilizadas e a metodologia de como foi concebido.

A Figura 9, demonstra a utilização das camadas utilizadas no projeto, onde a camada de Apresentação comunica somente com a camada de Controle, a qual fornece as informações necessárias e interagem com a camada de Negócios e para finalizar essa camada chama a camada de Persistência e acessa o banco de dados.

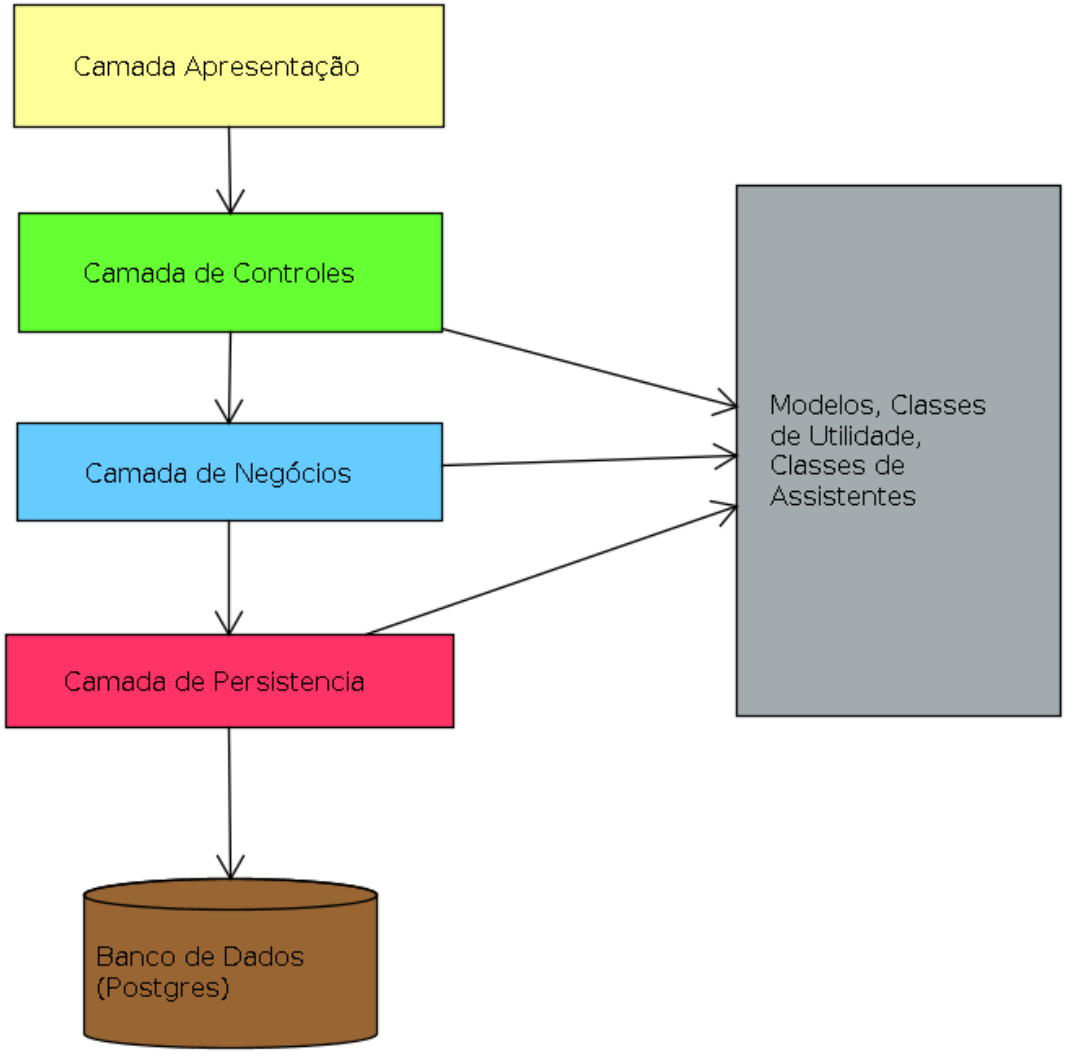

Figura 9: Demonstração do diagrama para a construção da arquitetura do projeto.

Essa arquitetura também pode ser conhecida como MVC Push ou *Action-Based (Baseado em ações)*, onde envia os dados para a camada de visão, com isso fornece um grande desacoplamento da camada de visão, mas pode ocorrer a perca da possibilidade de temos componentes ricos controlados diretamente pelo Java. Diferente do que ocorre no JSF (JavaServer Faces), que é uma tecnologia Pull também conhecida por *Component-Based (Baseado em componentes)*, onde a camada de visão exerce um alto acoplamento com a camada de controle, que pode gerar alguns problemas, mas se ganha a possibilidade de criar componentes ricos controlados pela linguagem. (ALMEIDA, 2012)

3.1. Sobre o desenvolvimento do projeto

A idealização do projeto Automagz, é para entrar no mercado de comercialização de veículos e serviços por meio de anúncios e fornecimento de consultas na internet.

O desenvolvimento do projeto começou com o levantamento dos requisitos necessários para a realização dos objetivos propostos para esse sistema. Estes objetivos são:

1. Manter informações dos usuários.
2. Manter informações dos anúncios dos usuários.
3. Autenticar e controlar os usuários e seus anúncios.
4. Fornecer uma página de consulta dessas informações
5. Fornecer uma página de consulta detalhada do anúncio.
6. Fornecer uma busca dos anúncios cadastrados.

Após definido os objetivos foi iniciado o mapeamento das entidades relacionadas utilizando a ferramente Astah Professional, ferramenta essa de modelagem UML (*Unified Modeling Language*) Linguagem Unificada de Modelagem. Na versão professional ela exige ser licenciada, mas a empresa que a mantêm fornece uma licença educacional para treinamento de estudantes.

Após criados os diagramas nesta ferramenta, os mesmos foram exportados através dela, após isso as classes foram importadas e utilizadas no

projeto que foi construído por meio da IDE (*Integrated Development Environment*) Ambiente integrado de desenvolvimento.

A IDE utilizada para o desenvolvimento desse projeto foi a Eclipse na versão Luna, a qual tem suporte ao Java 8, Tomcat 8, mas o projeto foi construido com Java 7 e o Tomcat 8.

Este projeto apresenta também como parte importante o armazenamento de suas informações, ocorrendo assim a persistência dos dados. Para termos esses dados armazenados utilizamos o banco de dados PostgreSQL na versão 9.3, este SGBD (Sistema de Gerenciamento de Banco de Dados) é fornecido gratuitamente por ser um projeto *open-source*.

O projeto foi construído utilizando o Apache Maven que é um projeto que serve para efetuar o gerenciamento e compressão do projeto. Utilizando essa ferramenta as dependências do projeto são gerenciadas por ela, fazendo com que o projeto seja mais fácil de ser mantido. Pois os .jar que farão parte da biblioteca do projeto ficarão associados ao arquivo *pom.xml* o que torna mais simples o gerenciamento, pois não é mantido o arquivo [arquivo].jar em específico.

Também fazendo com que os projetos que utilizam dessa tecnologia fiquem de tamanho reduzido.

Foi adotada a utilização do versionamento do código gerado para este projeto, a ferramenta selecionada foi o GIT, iniciamos o versionamento sendo utilizado o Github para gravar seu código, mas durante o processo alteramos o armazenamento para o repositório Bitbucket, ambos gratuitos mas com uma grande diferença os projetos mantidos no Github são públicos, enquanto os projetos no Bitbucket são privados e gratuitos até 5 desenvolvedores.

Por ser um projeto particular e ser comercializável, realizei essa alteração para que não aja cópias não autorizadas do projeto.

3.2. Diagrama de Caso de uso do serviço de Personal Car.

A Figura 10, apresenta o diagrama de uso de um serviço que será fornecido pelo portal. Esse serviço envolve consultoria de serviços automotivos. Onde será analisado o problema que o cliente tem e o indicaremos para os melhores profissionais. Além de trabalhar-mos o orçamento do cliente para que tenha o melhor serviço pelo menor preço.

Essa comodidade trará benefícios, como o conforto e resultados expressivos nos valores da execução de serviços.

O ator Pessoa deve primeiramente realizar o cadastro no portal, depois seu cadastro é validado por uma mensagem que chegará em seu *e-mail* (correio eletrônico). Feito essa ação o ator poderá realizar a autenticação e solicitar seu atendimento.

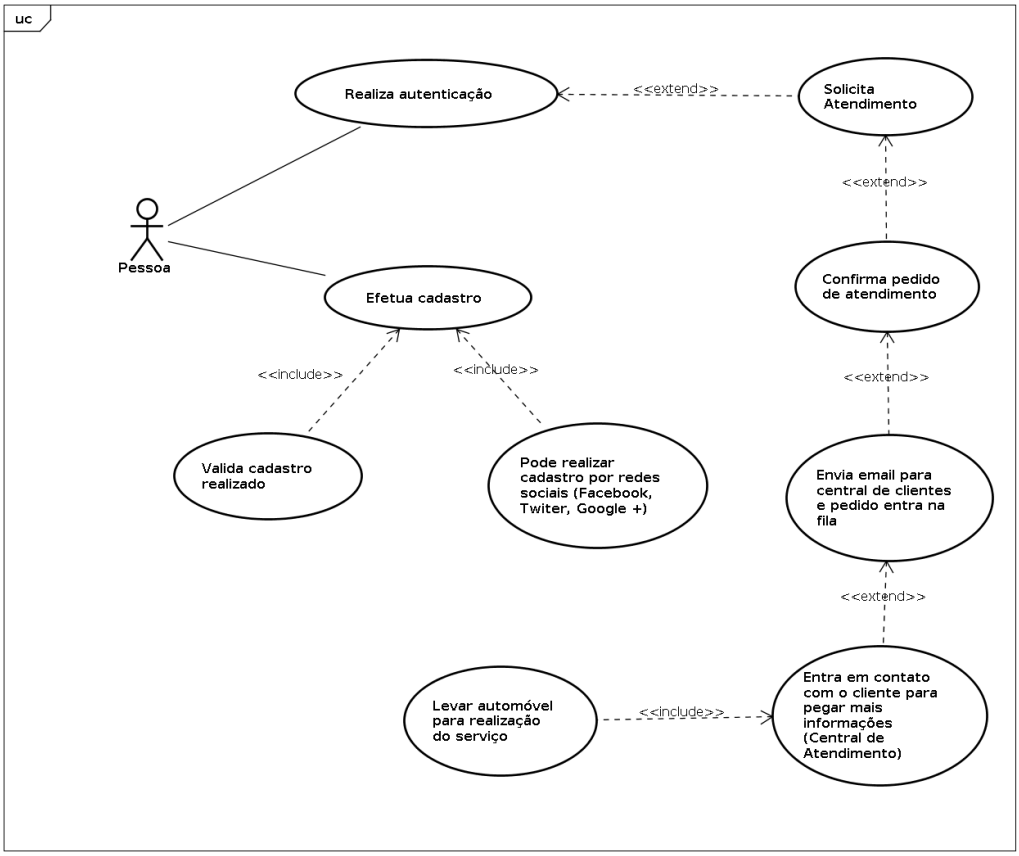

Figura 10: Diagrama do Caso de Uso do serviço de Personal Car.

3.4. Diagrama de classe para o cadastro de cliente

O diagrama de classes de cadastro de cliente tem a função de demonstrar quais foram as entidades envolvida.

A Figura 11, mostra todas classes que foram criadas para a construção do projeto e que estão diretamente envolvidas, com a classe de Cliente.

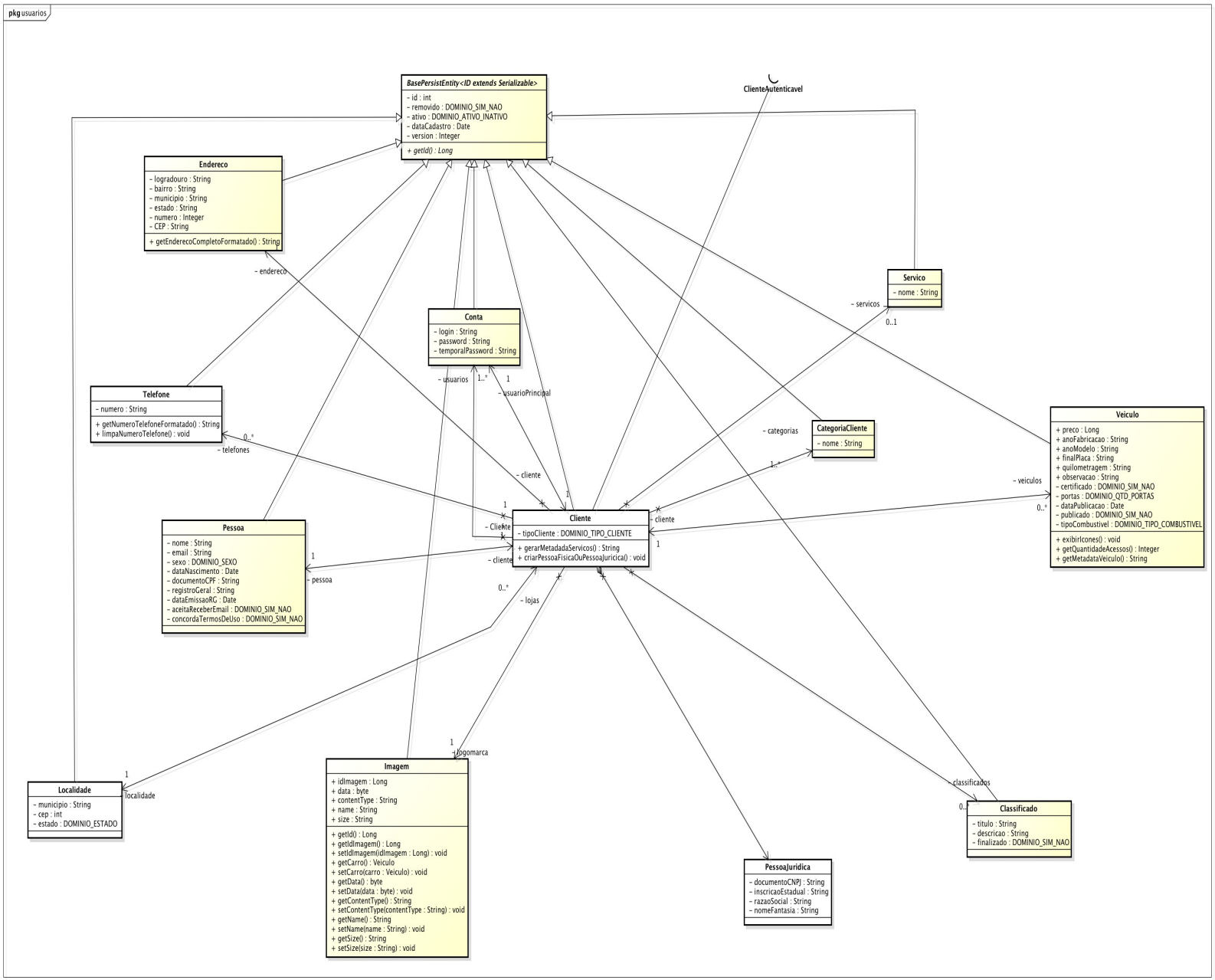

Figura 11: Diagrama das classes que estão envolvidas com o cadastro de cliente.

Foi criado uma classe principal BasePersistentEntity, que mantém informações relevantes e utilizadas em todas as outras entidades. As classes que herdam de BasePersistentEntity trará implementado atributos como, id, removido, ativo e data cadastro.

O mapeamento de classes para a criação da entidade de Cliente envolvem diversas outras entidades que a compõem. Dentre essas entidades podemos citar: Conta, Endereço, Telefone, Serviços, Categoria de Clientes, Informações Pessoais (Pessoa), Informações Jurídicas (Pessoa Jurídica), Classificados, Veículos e a seu Avatar (imagem pessoal).

3.4. Diagrama de classe para o cadastro de veículo

A Figura 12 representa o diagrama que demonstra quais atributos estão presentes na classe Veículo, e suas agregações com todas as outras classes que são utilizadas para montar a sua composição. É importante lembrar que um veículo pertence a um cliente, pode ter muitas imagens, uma cor, marca, modelo e seus itens opcionais além de demonstrar as condições que o veículo está.

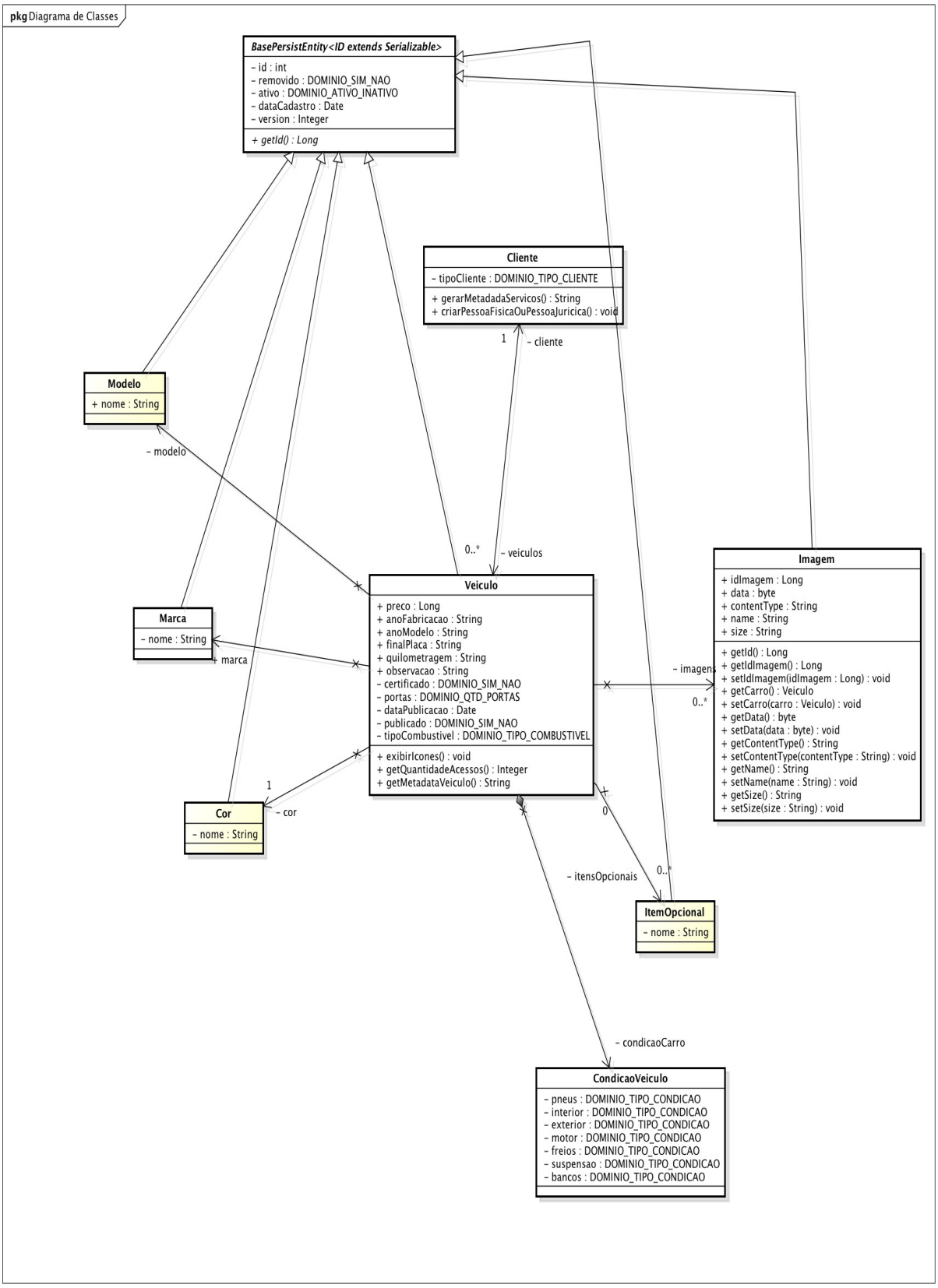

Figura 12: Diagrama de classes das classes envolvidas com um veículo

3.5. Código gerado no projeto

Foi adicionado no Apêndice 1 alguns exemplos de código que foram gerados para o projeto.

Foram adicionadas as clases GenericDAO.java, VeiculoDAO.java, ClienteController.java, ClienteValidator.java e um arquivo da camada de visualização Index.jsp.

Esses arquivos que são responsáveis pela Apresentação, Controle, Validação dos dados e Persistência das informações.

3.6. A escalabilidade no projeto

Como o sistema ainda está pequeno e em desenvolvimento não conhecemos os gargalos que poderão ocorrer.

Mas já pensamos algumas soluções que poderão ser utilizadas. No projeto conforme o item Escalabilidade.

Como tomada de ação, foi pensado adicionar o serviço de Cache já na página principal, também fazer Cache dos resultados das consultas mais solicitadas.

Colocar solicitações das páginas para que sejam executadas em *Client-Side* (Lado do cliente) em vez de *Server-Side* (Lado do servidor), como passar as imagens em base64 essa técnica é chamada de *image-inline* (imagem em linha) e deixar o navegador do cliente resolver as imagens, diminuindo a quantidade de requisições que são feitas para o servidor. E utilizar conexões *Stateless*, passando para o cliente o resultado das páginas em um formato texto como o Json.

Também para aumentar a escalabilidade foi utilizado o serviço da empresa Cloudinary, efetuando a hospedagem das imagens externamente ao servidor principal.

3.7. Arquitetura do Projeto

A Figura 13, demonstra a arquitetura geral do sistema criado e da sua concepção. As partes compostas desse informações da utilização do projeto são as camadas Vraptor que mantém as diversas funcionalidades do framework. E

para um complemento de algumas funcionalidades que não estão presentes no Vraptor, foi criado o Paiter-Core.

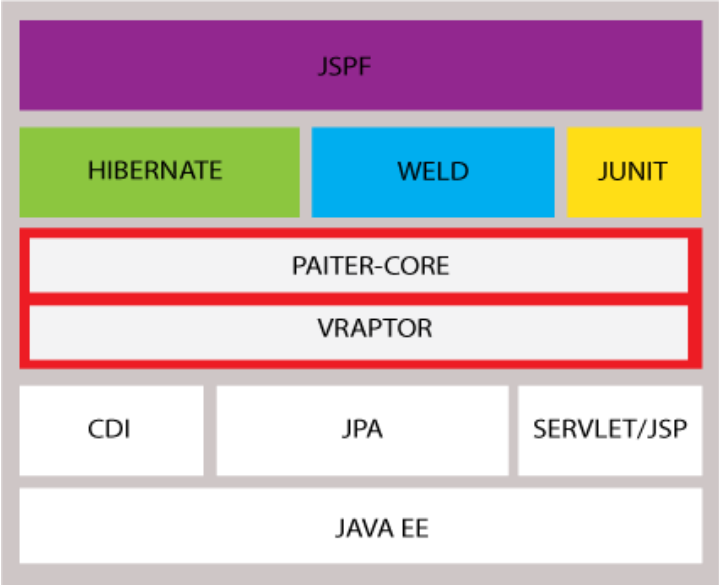

Figura 13: Demonstra a arquitetura em camadas utilizadas no projeto.

3.8. Aplicação de testes a arquitetura proposta

Aplicando-se testes com a ferramenta JMetter, temos o objetivo de exibir que esta arquitetura reponde bem aos desafios propostos.

Para aferir a capacidade, a disponibilidade em casos extremos ou apenas verificar o tempo de resposta de uma determina funcionalidade da aplicação, se faz necessário a utilização de ferramentas que permitam a simulação do uso e a devida documentação dos resultados obtidos.

Na Figura 14, constata-se as configurações utilizadas para os testes, foi criada um *Thread Group* de 500 (quinhentos) usuários que acessaram entre eles um período de 0.2 milissegundos fazendo esse processo por 1 vez.

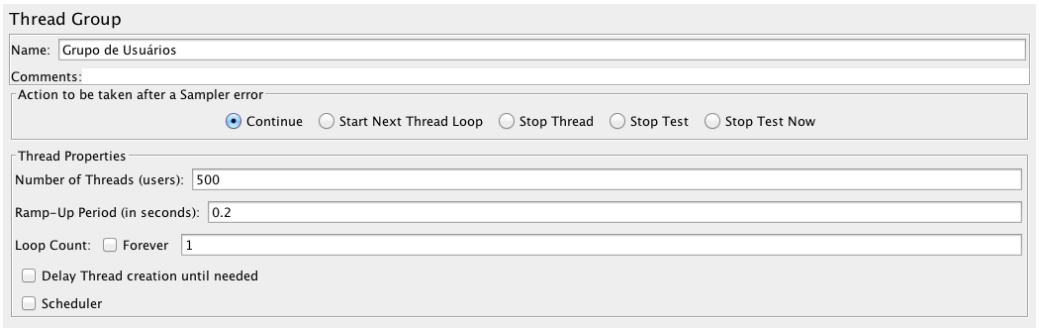

Figura 14: Configuração do Thread Group, onde é definido o número de usuários que acessarão a aplicação

O comportamento da aplicação pode ser analisado, também, de maneira gráfica em tempo de execução. Para isto, usamos o elemento *Graph Results*, a Figura 15 que apresenta os resultados para os parâmetros informado na Figura 14:

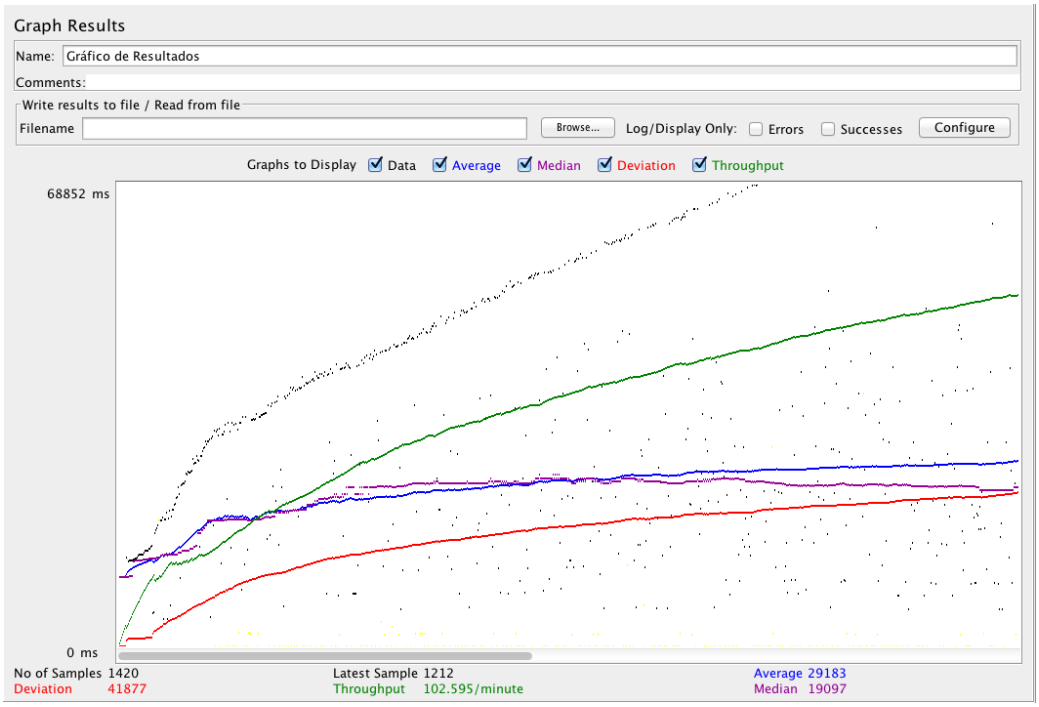

Figura 15: Resultado gráficos da aplicação dos testes

- *Average* - Média entre o tempo e o número de requisições;
- *Median* - Mediana é um valor que divide as amostras em duas partes iguais. Metade das amostras são menores que a média e a outra metade maior que a média, podendo ter algumas amostras com valor igual a média;

- *Deviation* - É a medida da variação de um dado conjunto de dados;
- *Throughput* - Número de amostra por unidade de tempo.

A Figura 16, se responsabiliza por demonstrar os resultados obtidos de cada teste realizado. Foram executadas 1409 requisições nos 3 testes e a página que ocorreu mais erros foi o item PesquisaVeiculo, onde apresenta uma taxa de erro de 52%.

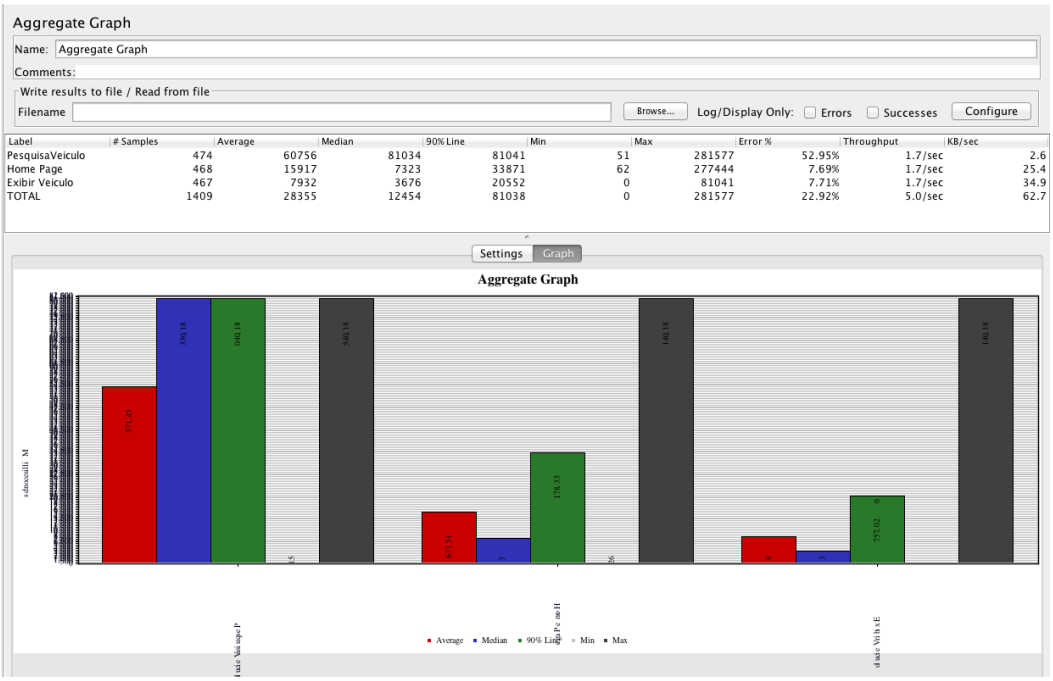

Figura 16: Resultado do Aggregate Graph para os testes propostos

3.9. Diagrama de Pacotes do Sistema

A Figura 17, demonstra o diagrama de pacotes criados para o sistema. Nela apresenta 3 *packages* (pacotes) importantes para o sistema, os pacotes automag, caelum e paiter.

O pacote automag, armazena informações referentes ao projeto como suas entidades, domínios, *controller* das páginas, utilitários e classes de negócio.

O pacote caelum, traz uma plugins que são utilizados diretamente como Vraptor.

E o pacote paiter, contém a abstração de serviços importantes para o sistema como Persistência, Segurança, *Controller*, Interceptadores, Paginadores, Entidades e outros.

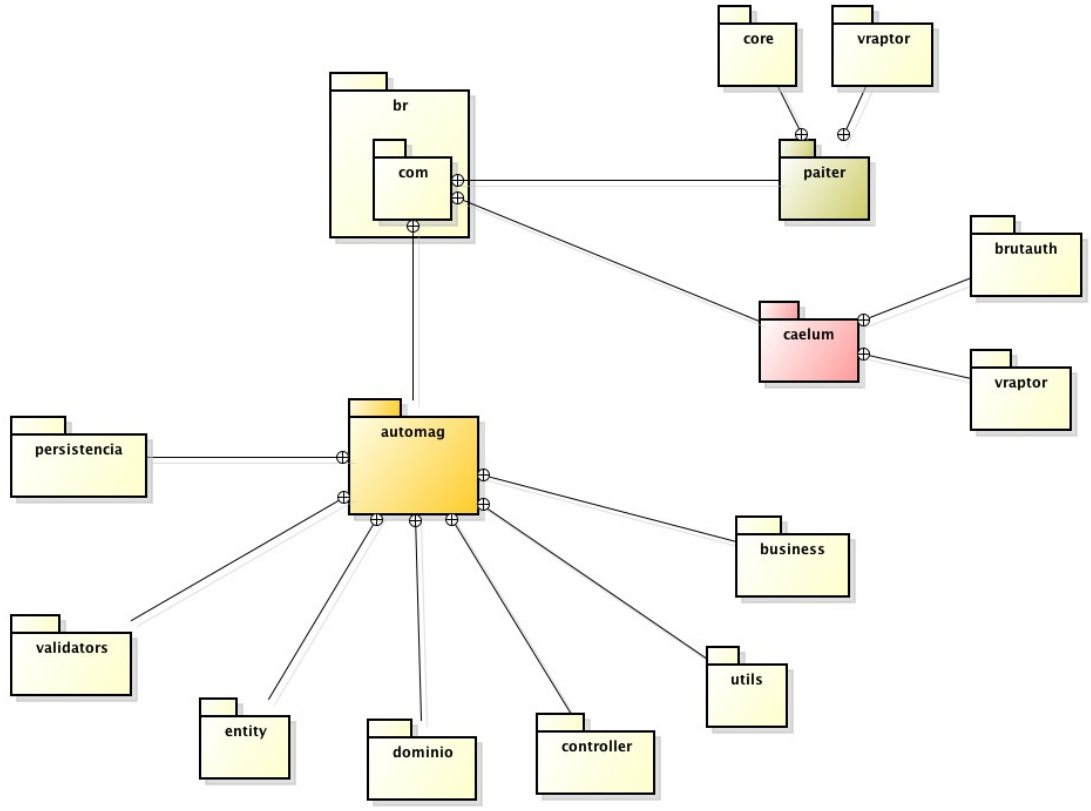

Figura 17: Diagrama de pacotes do sistema

A utilização dessa disposição de pacotes e seus conteúdos, dá à aplicação a coesão e baixo acoplamento dos serviços dentro das classes.

4. RESULTADOS

4.1. Página de Front-End

A Figura 18 apresenta a página principal do portal, página de Front-End, é composta por campos de autenticação, botão de criação de nova conta, exibir os últimos anúncios em destaque, últimas ofertas cadastradas. Algumas funcionalidades ainda não foram finalizadas ou até mesmo traduzidas, mas isso será alvo das próximas publicações do projeto.

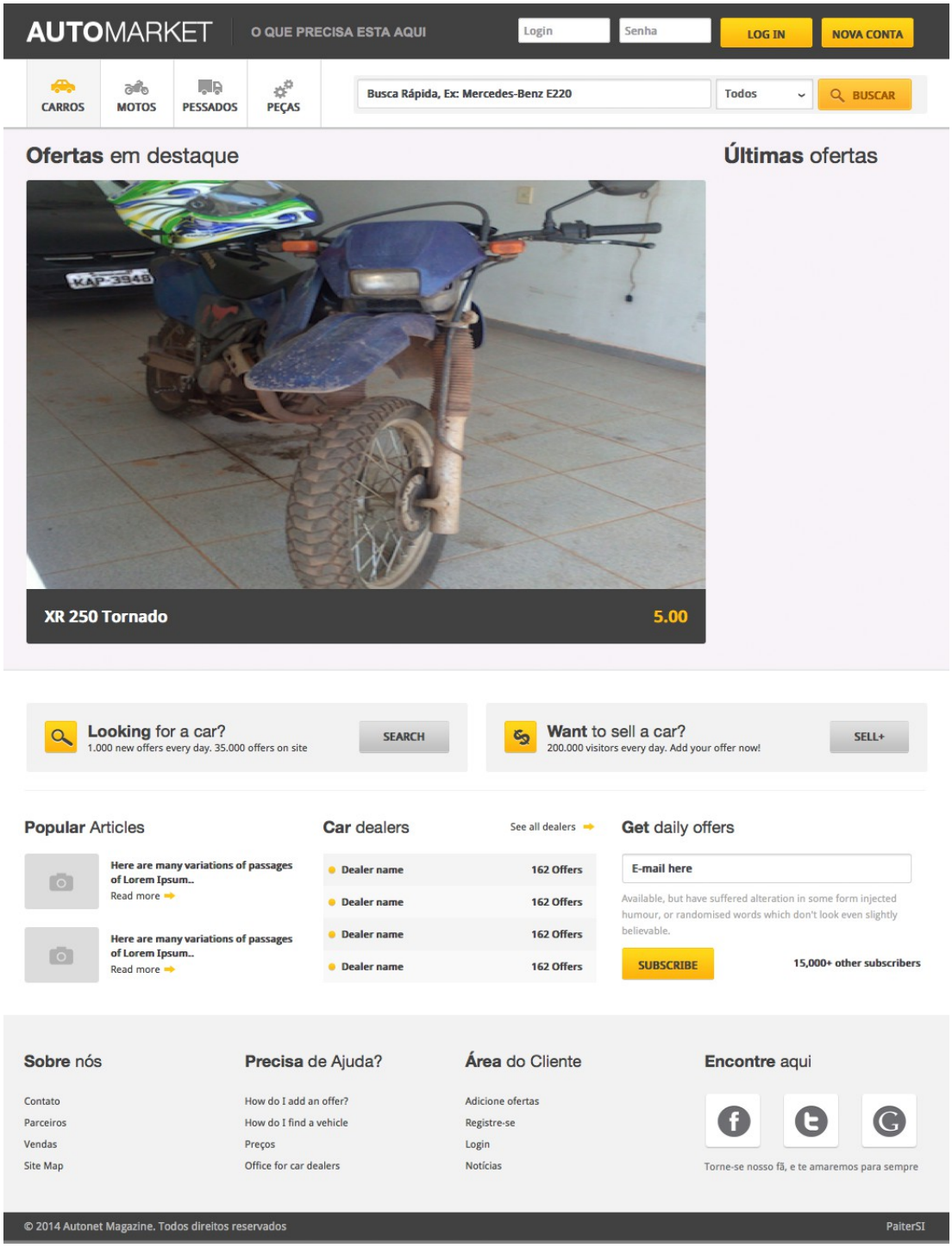

Figura 18: Tela de Front-End, capa do portal www.automagz.com.br.

4.2. Página de cadastro

A Figura 19 demonstra a página de cadastro de usuários e identificação de novo usuário. Estes são recursos que são necessários para o cadastro inicial que solicitará para inserir o nome, e-mail, login e uma senha que deve ser verificada se estão idênticas. Também é a página que é direcionada caso usuário informe um usuário e senha inválidos.

Também está presente um formulário para a autenticação do usuário, não necessitando fazer toda a autenticação somente no cabeçalho da página.

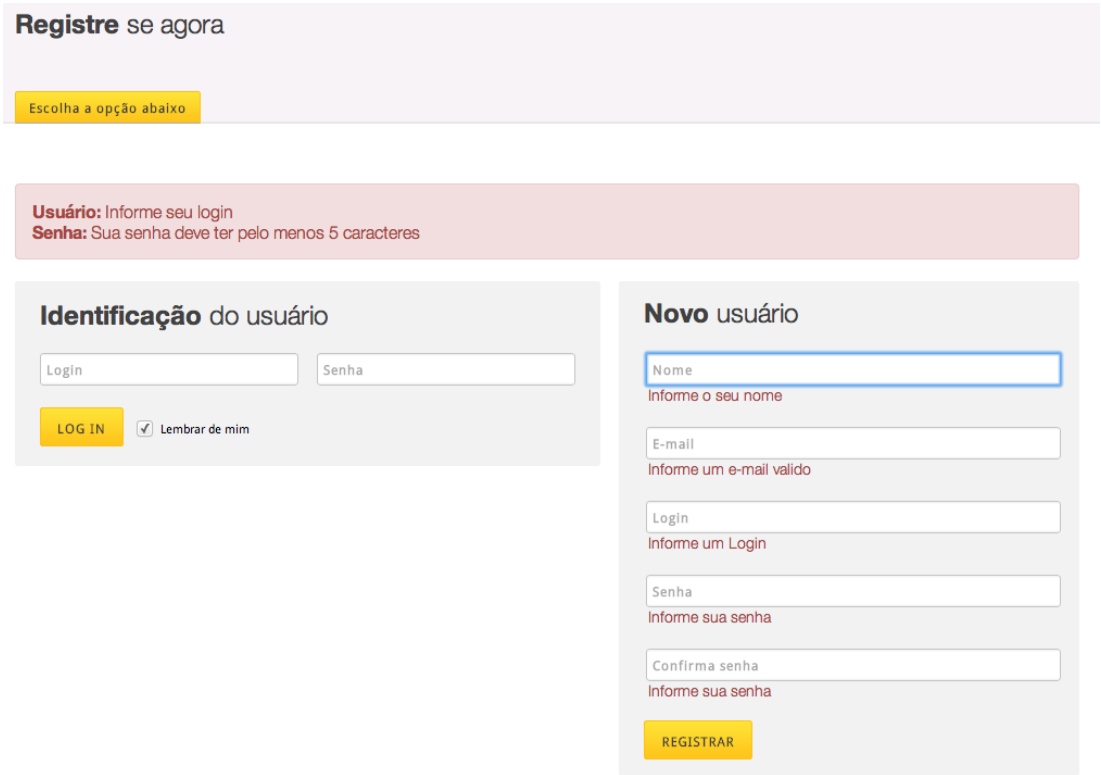

Figura 19: Tela de cadastro e autenticação de usuário do site.

4.3. Página de resultado da consulta.

A Figura 20 apresenta a página de consulta de veículos. O internauta poderá pesquisar tanto pelo modelo quanto pelo nome do modelo, quanto pelo nome da marca e será exibido o resultado dos veículos que estão com as características informadas.

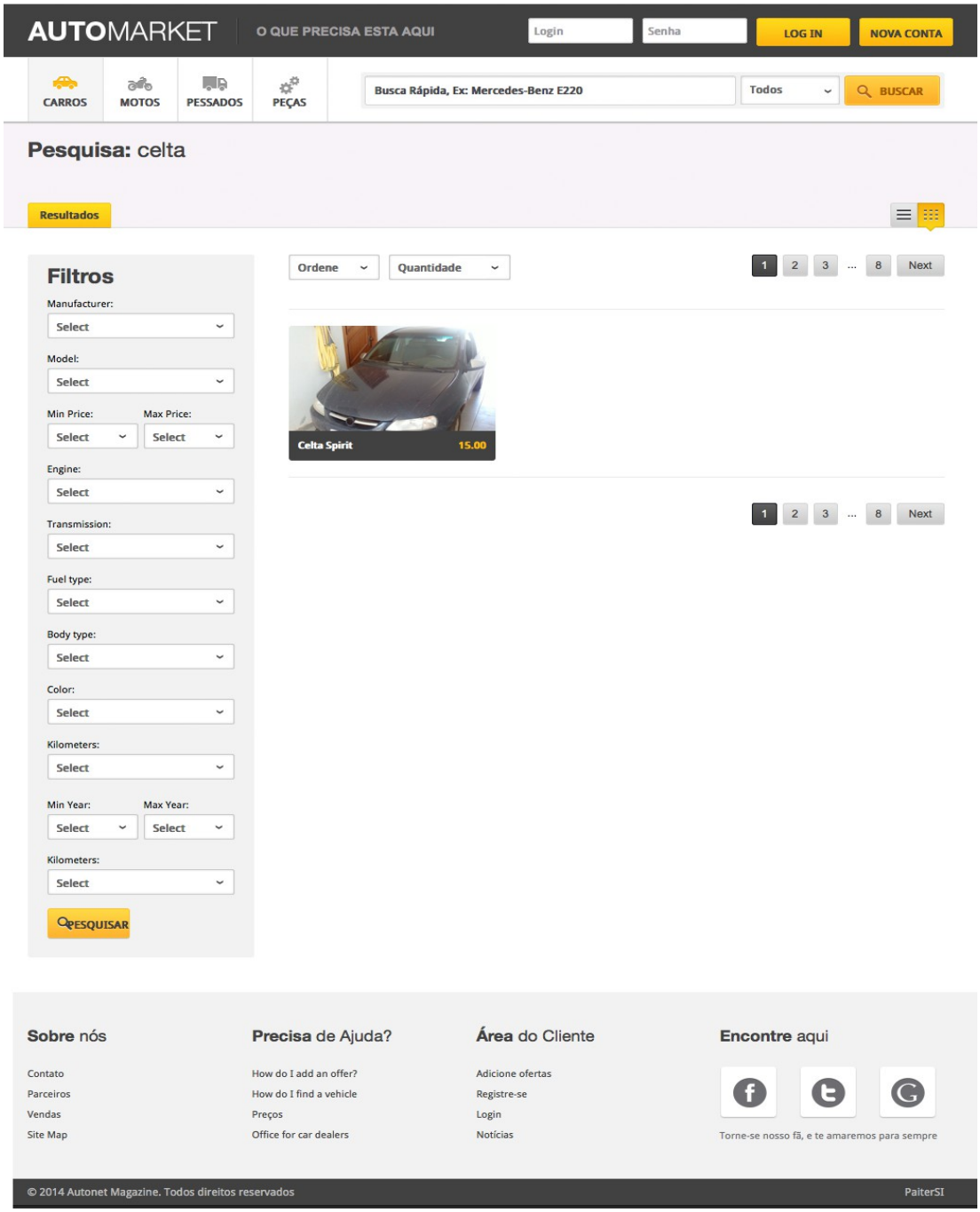

Figura 20: Tela de apresentação do resultado da pesquisa.

Ainda fazendo parte da pesquisa a Figura 21 apresenta as funcionalidades dos filtros laterais, que melhorarão a pesquisa do cliente o portal. Encontrando assim o veículo que ele quer se estiver em nossa base.

Também apresenta uma variação na forma de apresentar o resultado das consultas. Quando o internauta clicar nesse botão ele altera a forma de exibição do conteúdo da página passando da exibição em quadros para uma linha com as informações iniciais do anúncio.

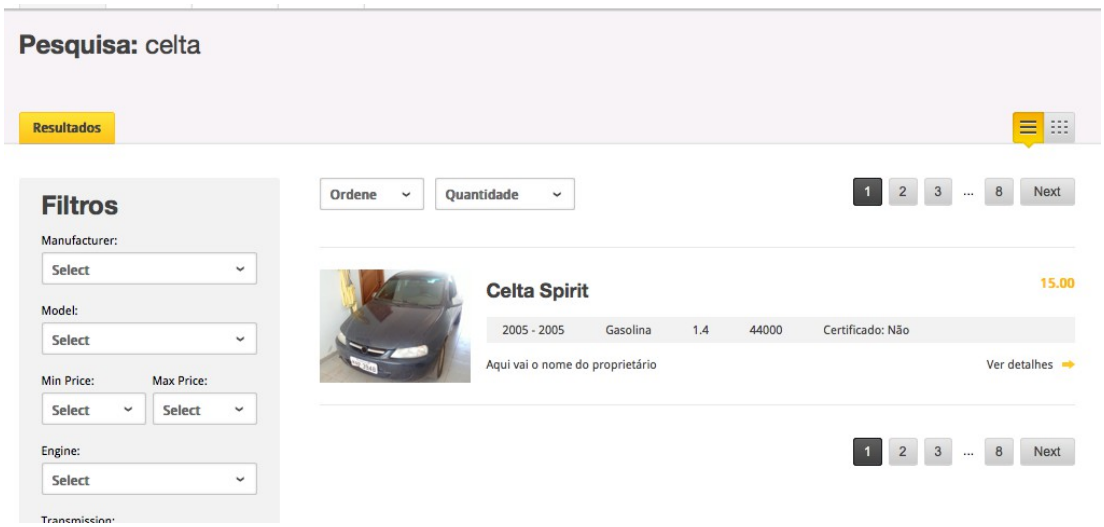

Figura 21: Está figura demonstra a alteração na forma de exibir o resultado da consulta.

4.4. Página de exibição do veículo

Está tela é uma das principais do sistema pois é aqui que os internautas analisarão informações do veículo selecionado. Na Figura 22, é exibida as funcionalidades que acompanharam esta página.

1. Galeria de fotos do veículo.

2. O botão como chegar será usado para mapear o endereço dos clientes do tipo pessoa jurídica, que mostrará o endereço do logista.

3. Botão de compartilhamento é um forma do cliente clicar e divulgar que está vendendo algum item e também fornecer um marketing de baixo custo anunciando em sua rede social.

4. Poderá imprimir o anúncio com todas as informações chegar até vendedor.

5. A ainda uma estratégia de entrar em contato com algumas empresas de financiamento para saber se existe o interesse em divulgar seus serviços e fornecer suas taxas para serem utilizadas no formulário Calculadora para saber possivelmente quanto pagará em seu financiamento.

6. Também será apresentado uma lista de veículos compatíveis com o que está visualizando no momento, do acesso.

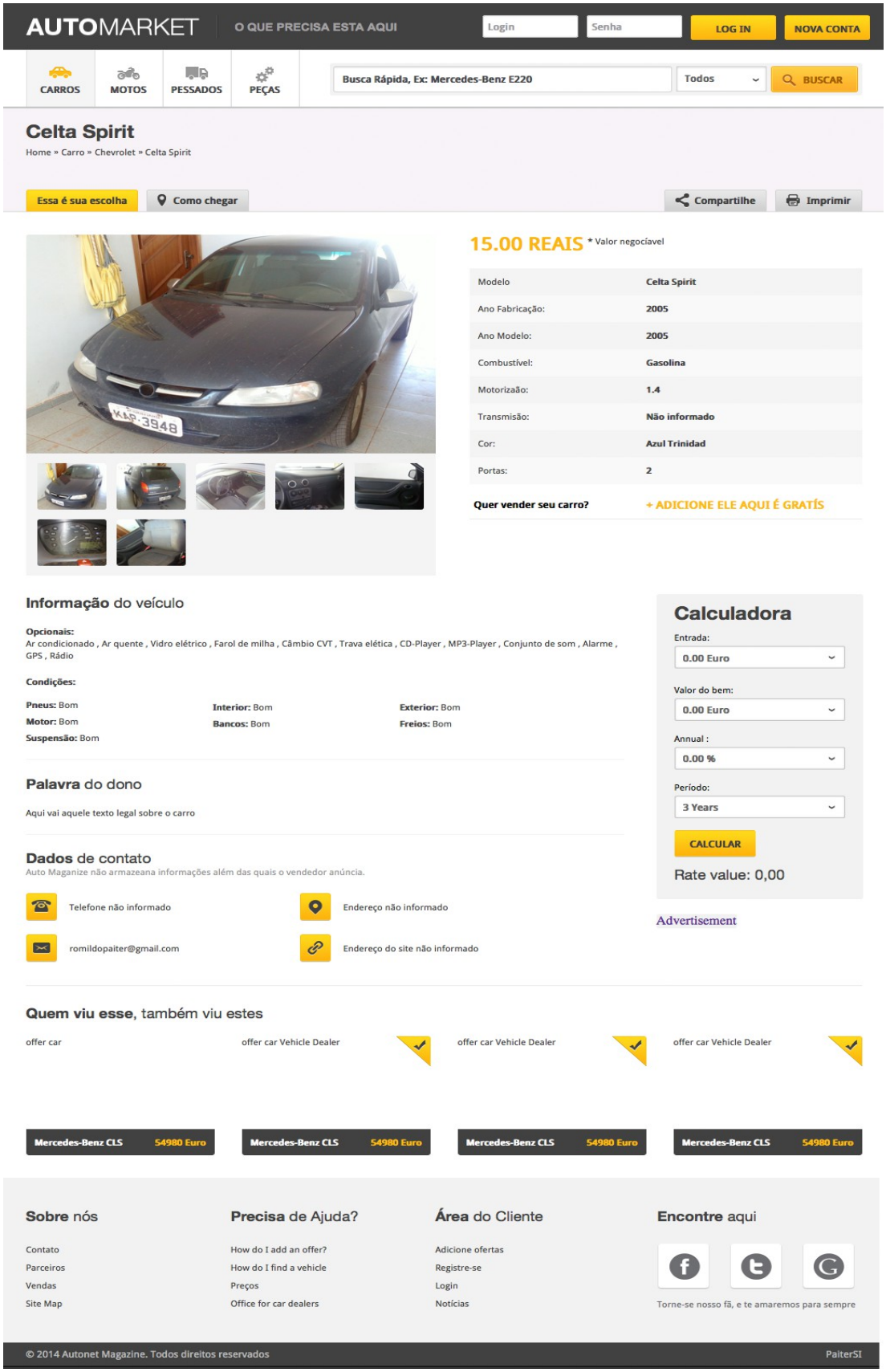

Figura 22: Página exibição do veículo selecionado

4.5. Página de dashboard do internauta

A Figura 23 mostra a página de dashboard que centraliza as opções que estarão disponíveis para o usuário. Durante o período de estágio não foi possível concluir todas essas opções. Mas já existe a demanda para a conclusão do conteúdo dessa página fornecendo para o internauta qualidade no conteúdo e também a aplicação de requisitos de interface.

Figura 23: Página de dashboard de acesso do usuário

4.6. Página de listagem de veículos

A Figura 24 apresenta o controle do cliente referente aos veículos que pertencem ao usuário no sistema. Nessa página será possível remover anúncios, finalizar, alterar, marcar como vendido.

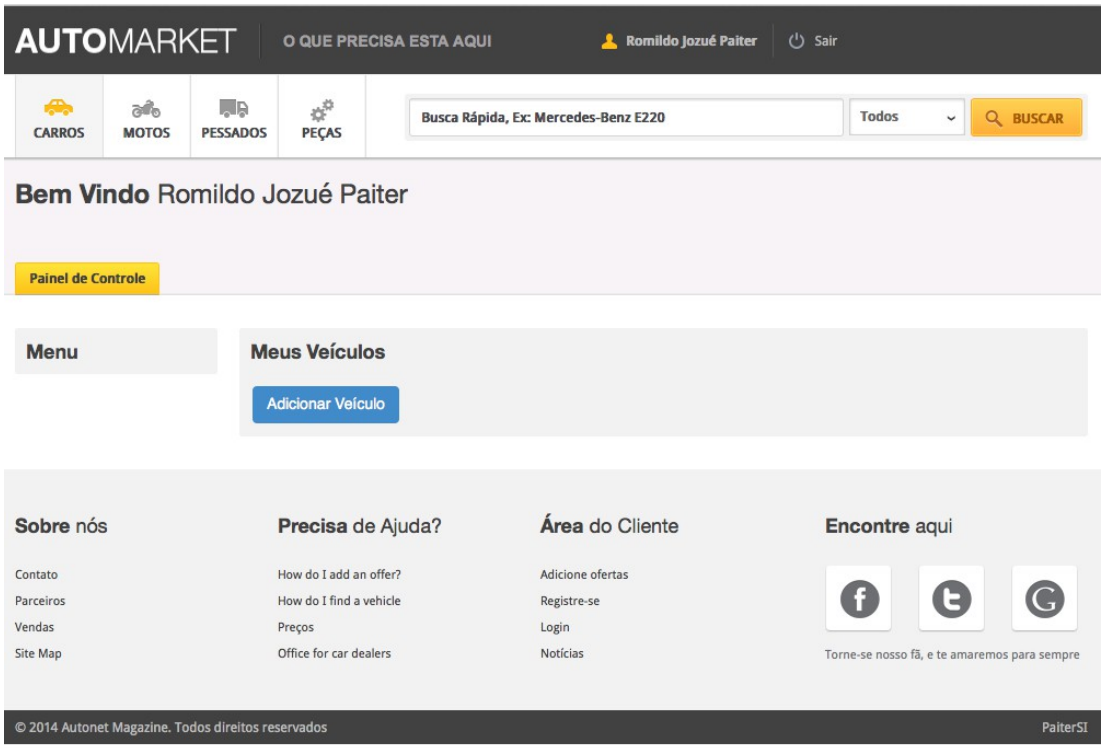

Figura 24: Página de inclusão e listagem de veículos dos clientes

4.7. Página de cadastro de veículos

A página de cadastro de veículos apresentada na Figura 25 tem alto grau de complexidade, pois apresenta diversas interações com banco de dados e também é responsável por controlar muitas informações sobre o anúncio que está sendo postado.

Os campos Marca e Modelo ficaram livre para entrada de conteúdo, mas já existe uma ferramenta de autocomplete que trará as marcas e modelos já existentes no banco. Essa opção foi escolhida por não ter uma base com todas essas informações, então essa decisão de manter o campo aberto para usuário informar. Estamos cientes dos possíveis problemas que podem ocorrer, mas inicialmente trataremos dessa maneira a aplicação.

O cadastro de veículo utiliza muitos enumerados para manter o padrão com valores fixos e também uma forma de padronizar os dados, que serão informados pelos clientes, proporcioando uma busca concisa.

Ainda ocorre a validação no momento do cadastro informando quais campos são obrigatórios e validação da entrada do dado no *controller*, para que não fique faltando informações.

Foi utilizado algumas interações em jQuery para efetuar a validação, para aumentar o número de imagens que estão sendo inseridas.

A Figura 25 (a,b) foi particionada para ser vista com qualidade.

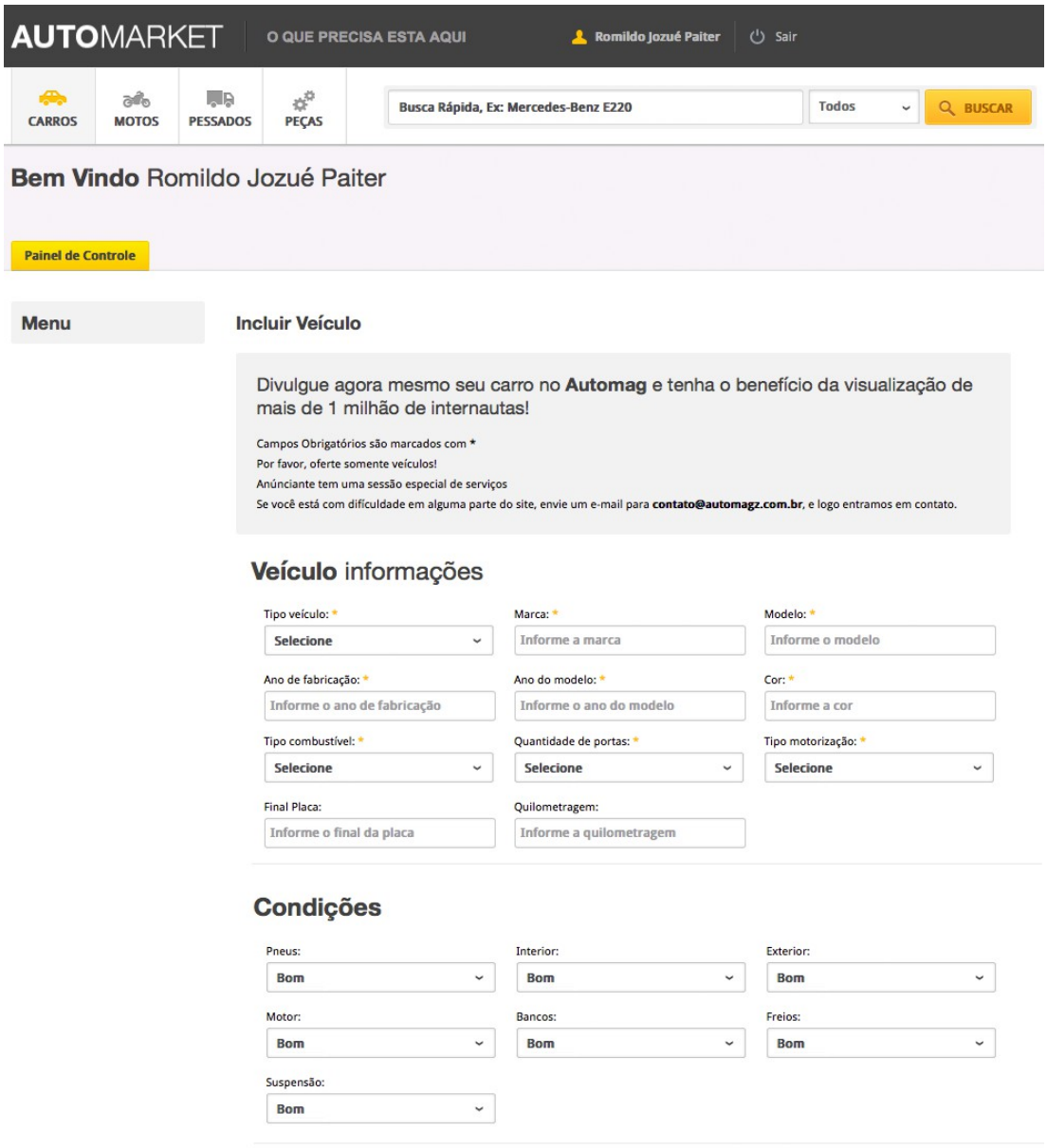

Figura 20.a Página de cadastro de veículos.

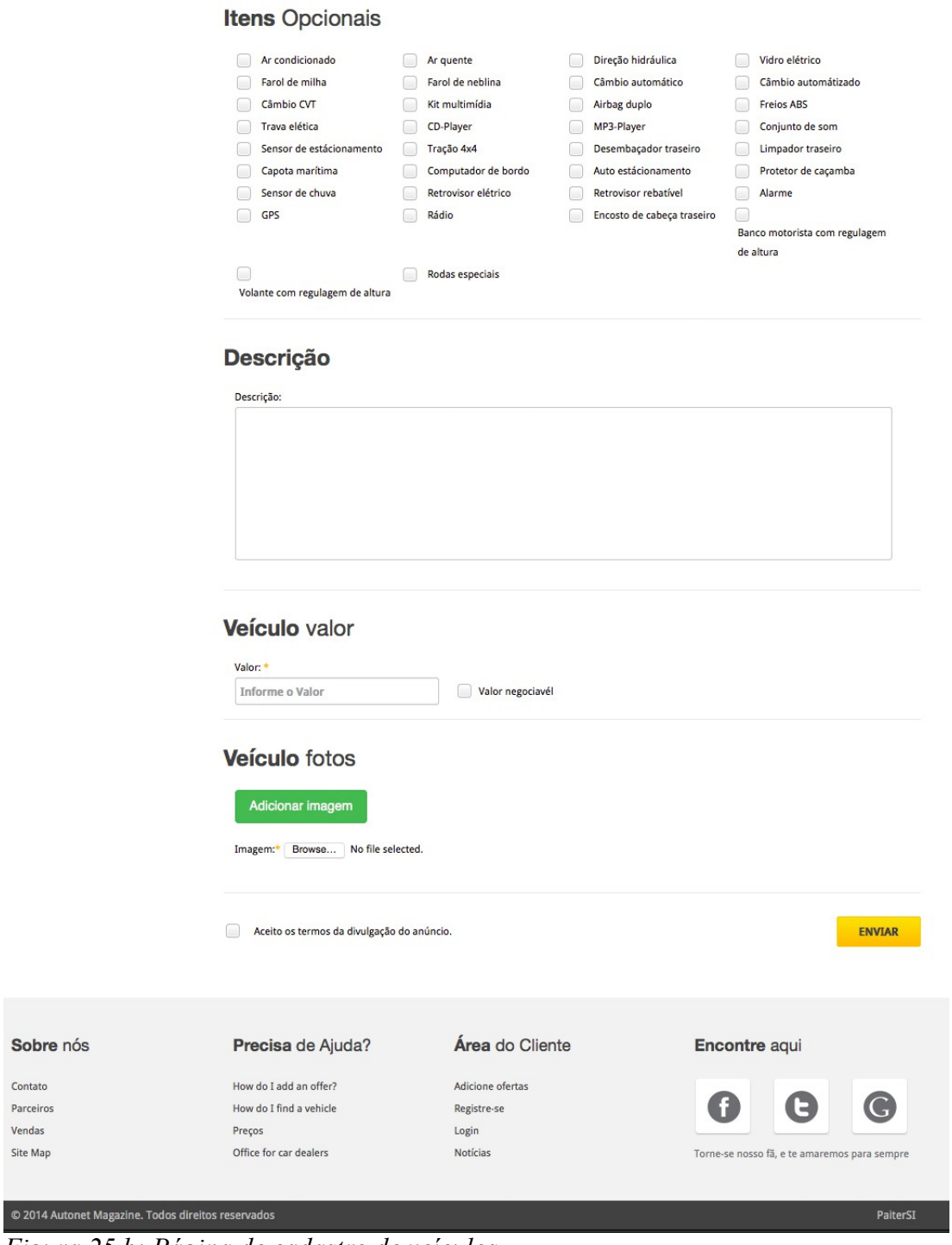

Figura 25.b: Página de cadastro de veículos.

4.8. Execução de teste de performance da aplicação

Para executar esses testes será utilizado o Apache JMetter. Onde vamos executar alguns testes e exibir os gráficos gerados.

5. DIFICULDADES ENCONTRADAS

Na construção desse trabalho podemos considerar que uma dificuldade, é a forma de integrar todo o conhecimento adquirido com as leituras.

O trabalho de arquiteto de software, é tomar decisões certas desde o início do projeto. Caso essas não ocorram podem gerar uma grande dificuldade de manutenção e evolução do projeto posteriormente.

A dificuldade acima relatada é sobre as atribuições que um arquiteto de software deve fazer. Mas a respeito da tecnologia utilizada o Vraptor, como toda vez que é vista pela primeira vez existe um tempo da curva de aprendizado, para contornar essa curva foi utilizado recursos como livros sobre Vraptor, fórum dos entusiastas, que por sinal é muito movimentado e tudo que perguntamos é recebido com bom grado e respondido. Endereço eletrônico do grupo, caelum-vraptor@googlegroups.com, outro detalhe importante sobre a comunidade que utiliza esse framework, pode ser buscado no GUJ (Grupo de Usuários Java).

Uma outra dificuldade que se tornou um conhecimento foi a utilização de componentes jQuery dentro do projeto, o que melhorou muito meu domínio dessa ferramenta.

6. CONCLUSÕES FINAIS

A utilização desse framework engrandeceu meu conhecimento, além também de facilitar o desenvolvimento desta aplicação, a tornando mais robusta. Também me dando um diferencial de competitividade no mercado de trabalho.

O Vraptor é um exclente framework e entrega o que promete, fornecendo os requisitos necessários para se ter um framework MVC, além de ter suporte a diversas tecnologias atuais do Java podendo ser utilizado em servidores mais simples com o Tomcat e conteiners mais avançados como o Wildfly 8.1.0 e Glassfish 4 que são conteiner EJB 3.0.

Como próximo passo continuarei com o projeto, e farei as adequações para atender os requisitos e colocar em produção na internet. Fornecendo um serviço diferenciado, que venha atender expectativas que serão criadas com o uso da ferramenta.

7. REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª Edição. Local São Paulo: Editora Pearson – Addison Wesley. 2011.

FOWLER, Martin. **Padrões de Arquitetura de Aplicações Corporativas**, Tradução Acauan Fernandes – Porto Alegre – Editora Bookman. 2006.

DENNIS, Alan / WIXON, Barbara. **Análise e Projeto de Sistemas**, 2ª Edição, Tradução Michele Geinhart – Rio de Janeiro – Editora LTC, 2011.

SILVEIRA, P. et al., **Introdução a Arquitetura e Design de Software: Uma visão sobre a plataforma JAVA**. 1ª Edição - São Paulo - Editora Campus. 2011.

CAVALCANTI, Lucas, **VRaptor, Desenvolvimento ágil para Web com JAVA**. 2ª Edição - São Paulo - Editora Casa do Código. 2014,

CORDEIRO, Gilliard, **CDI: Integre as dependências e contextos do seu código Java**. 1ª Edição - São Paulo - Editora Casa do Código. 2013.

JOHNSON, Rod et al. **Java Development with the Spring Framework** - Wrox Press Ltd, 2005.

JOHNSON, Rod, **Export one-to-one J2EE Design and Development. Fisrt Printed October** - Wrox Press Ltd, 2002.

GONCALVES, Edson. **Desenvolvendo aplicações web com JSP, Servlets, Java Server Faces, Hibernate, EJB3 Persistence e Ajax**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2007.

SINGH, I. et al. **Designing Enterprise Applications with the J2EE Platform**. Second Edition. 2nd ed. New Jersey, USA: Addison-Wesley, 2002.

COULOURIS, George et al. **Sistemas Distribuídos: Conceitos e Projeto** – 5 Edição – Porto Alegre - Bookman Editora, 2013

KING, Galvin; BAUER, Chistian. **Java Persistence com Hibernate**, Traduzido por: Gustavo Vinocur. - Rio de Janeiro – Editora Ciência Moderna, 2007.

ALEXANDER, C., ISHIKAWA S., SILVERSTAIN M., et al. **A Pattern Language:**

Towns, Buildings, Construction. Oxford University Press, 1977.

SHAW Mary, GARLAN, David; 1996, Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

ALMEIDA, Adriano. 2012, Entenda os MVCs e os frameworks Action e Component Based. Disponível em <http://blog.caelum.com.br/entenda-os-mvcs-e-os-frameworks-action-e-component-based/> Acesso em: 3 ago. 2014.

DELICATO, Flavia, 2008. Apresentação Arquitetura de Sistemas, Disponível em <http://www.dimap.ufrn.br/~flavia.delicato/ArquiteturaSistema2008_Completa.pdf> Acesso em: 3 ago. 2014.

MACHADO, Erich. jul 2008. Inversão de Controle, Disponível em: <http://blog.locaweb.com.br/%20tecnologia/inversao-de-controle/>. Acesso em: 05 ago 2014.

K19 Treinamentos, "Design Patterns em Java", Disponível em: http://www.k19.com.br/downloads/apostilas/java/k19-k51-design-patterns-em-java. Acesso em: 24 ago 2014.

XAVIER, Kleber. 2013 , "O que é arquitetura de software?"**.** Globalcode. Disponível em <http://blog.globalcode.com.br/2013/11/o-que-e-arquitetura-de-software.html>. Acesso em: 3 ago 2014.

Theme Forest. Disponível em <themeforest.net>. Acesso em: 07 ago. 2014.

CARVALHO, Raphael. Bizstart. Disponível em <http://bizstart.com.br/blog/bate-papo-sobre-startups/escalabilidade-o-que-e-para-que-serve-por-que-ter-1858>. Acesso em: 11 ago. 2014.

Astah. Disponível em <http://astah.net/>. Acesso em: 11 ago. 2014.

Bitbucket. Disponível em <https://bitbucket.org/>. Acesso em: 11 ago. 2014.

PostgreSql. Disponível em <http://www.postgresql.org/>. Acesso em: 11 ago. 2014.

URUBATAN, Rodrigo. Disponível em <http://sobrecodigo.com/os-4-segredos-para-um-sistema-altamente-escalavel/>. Acesso em: 11 ago. 2014.

Eclipse. Disponível em <https://www.eclipse.org/>. Acesso em: 11 ago. 2014.

Apache Maven. Disponível em <http://maven.apache.org/>. Acesso em: 09 ago. 2014.

Hibernate, Disponível em <http://hibernate.org/validator/documentation/getting-started/>. Acessado em: 10 ago. 2014.

Cass, Stephen; Diakopoulos, Nick; Romero Joshua J. 2014, IEEE Spectrum. Disponível em

<http://spectrum.ieee.org/static/interactive-the-top-programming-languages#index/1/0/0/0/1/50/1/50/1/50/1/30/1/30/1/30/1/20/1/20/1/5/1/5/1/20/1/100/>. Acesso em: 07 ago. 2014.

APÊNDICE 1

APÊNDICE A - Algumas classes que foram implementas e utilizadas no projeto.

Classe GenericDAO.java

```
package br.com.paiter.core.persistencia.impl;


public class GenericDAO<PK, T> {

	private EntityManager entityManager;

	@Inject
	public GenericDAO(EntityManager entityManager) {
		this.entityManager = entityManager;
	}

	@SuppressWarnings("unchecked")
	public T getById(PK pk) {
		return (T) entityManager.find(getTypeClass(), pk);
	}

	public void save(T entity) throws Exception {
		try {
			entityManager.persist(entity);
		} catch (Exception e) {
			e.printStackTrace();
			throw new Exception(e.getMessage());
		}
	}

	public T update(T entity) {
		entityManager.clear();
		return entityManager.merge(entity);
	}

	public void delete(T entity) {
		entityManager.remove(entity);
	}

	@SuppressWarnings("unchecked")
	public List<T> findAll() {
		return entityManager.createQuery(("FROM " + getTypeClass().getName()))
				.getResultList();
	}

	private Class<?> getTypeClass() {
		Class<?> clazz = (Class<?>) ((ParameterizedType) this.getClass()
				.getGenericSuperclass()).getActualTypeArguments()[1];
		return clazz;
	}

	public EntityManager getEntityManager() {
		return entityManager;
	}

}
```

Classe VeiculoDAO.java

```
package br.com.automag.persistencia;


public class VeiculoDAO extends GenericDAO<Long, Veiculo> implements Serializable {

	int pageNumber = 1;
	int pageSize = 10;

	@Inject
```

```
	public VeiculoDAO(EntityManager entityManager) {
		super(entityManager);
	}

	@SuppressWarnings("unchecked")
	public List<Veiculo> findVeiculosEmDestaqueIncluidos(){
		Session session = this.getEntityManager().unwrap(Session.class);

		Criteria criteria = session.createCriteria(Veiculo.class, "vei");
		criteria.setMaxResults(3);
		criteria.setFetchMode("vei.modelo", FetchMode.JOIN);
		criteria.createAlias("vei.modelo", "mod");
		criteria.addOrder(Order.desc("vei.id"));

		return (List<Veiculo>) criteria.list();
	}

	@SuppressWarnings("unchecked")
	public List<Veiculo> findUltimosVeiculosIncluidos(){
		Session session = this.getEntityManager().unwrap(Session.class);

		Criteria criteria = session.createCriteria(Veiculo.class, "vei");
		criteria.setFirstResult(3);
		criteria.setMaxResults(5);
		criteria.setFetchMode("vei.modelo", FetchMode.JOIN);
		criteria.createAlias("vei.modelo", "mod");
		criteria.addOrder(Order.desc("vei.id"));

		return (List<Veiculo>) criteria.list();
	}


	public Veiculo findVeiculoCompletoById(Long idVeiculo){
		return this.getEntityManager().createNamedQuery("Veiculo.buscarCompletoByIdVeiculo",
Veiculo.class)
				.setParameter("idVeiculo", idVeiculo)
				.getSingleResult();
	}

	@SuppressWarnings("unchecked")
	public List<Veiculo> findVeiculosByNomeModeloCriteria(String nomeModelo, String tipo_veiculo, int
first, int pageSize){

		Session session = this.getEntityManager().unwrap(Session.class);

		Criteria criteria = session.createCriteria(Veiculo.class, "vei");
		criteria.setFetchMode("vei.marca", FetchMode.JOIN);
		criteria.createAlias("vei.marca", "mar");
		criteria.setFetchMode("vei.modelo", FetchMode.JOIN);
		criteria.createAlias("vei.modelo", "mod");

		if(!tipo_veiculo.equals("TODOS")){
			criteria.add(Restrictions.eq("tipoVeiculo",
DOMINIO_TIPO_VEICULO.valueOf(tipo_veiculo)));
		}

		if(!StringUtils.isBlank(nomeModelo)){
			criteria.add(Restrictions.or(Restrictions.ilike("mod.nome", nomeModelo,
MatchMode.ANYWHERE),
					Restrictions.ilike("mar.nome", nomeModelo, MatchMode.ANYWHERE)));
		}

		criteria.setFirstResult((pageNumber-1) * pageSize);
		criteria.setMaxResults(pageSize);

		return (List<Veiculo>) criteria.list();
	}

	@SuppressWarnings("unchecked")
	public List<Veiculo> findVeiculosAllPagination(int first, int pageSize){
		return this.getEntityManager().createNamedQuery("Veiculo.buscarTodos")
						.setFirstResult(first)
						.setMaxResults(pageSize)
						.getResultList();
	}

	@SuppressWarnings("unchecked")
	public List<Veiculo> findVeiculosByNomeModeloPagination(String nomeModelo, int first, int
pageSize){
		return this.getEntityManager().createNamedQuery("Veiculo.buscarByNomeModelo")
				.setParameter("nomeModelo", nomeModelo)
```

```
                                .setFirstResult(first)
                                .setMaxResults(pageSize)
                                .getResultList();
        }

        @SuppressWarnings("unchecked")
        public List<Veiculo> findVeiculosByNomeModeloAndTipoVeiculoPagination(String nomeModelo, String
tipoVeiculo, int first, int pageSize){
                return
this.getEntityManager().createNamedQuery("Veiculo.buscarByNomeModeloAndTipoVeiculo")
                                .setParameter("nomeModelo", nomeModelo)
                                .setParameter("tipoVeiculo", tipoVeiculo)
                                .setFirstResult(first)
                                .setMaxResults(pageSize)
                                .getResultList();
        }

        public Long encontrarQuantidadeDeCarros() {
                return this.getEntityManager().createQuery("select count(v) from Veiculo v",
Long.class).getSingleResult();
        }
}
```

Classe ClienteController.java

```
package br.com.automag.controller;


@Routed
@Controller
@ConversationScoped
public class ClienteController implements Serializable {

        private static final long serialVersionUID = 1L;

        @Inject private Result result;
        @Inject private ClienteDAO clienteDAO;
        @Inject private VeiculoDAO veiculoDAO;
        @Inject private MarcaDAO   marcaDAO;
        @Inject private ModeloDAO  modeloDAO;
        @Inject private CorDAO     corDAO;
        @Inject private ItemOpcionalDAO itemOpcionalDAO;
        @Inject private HttpServletRequest request;
        @Inject private VeiculoValidator veiculoValidator;
        @Inject private Validator validator;
        @Inject private Conversation conversation;
        @Inject private UsuarioLogado usuarioLogado;


        @Get
        @CustomBrutauthRules(LogadoRule.class)
        public void dashboard(){

        }

        @Get
        @CustomBrutauthRules(LogadoRule.class)
        public void meusveiculos(){

        }

        @Get
        @CustomBrutauthRules(LogadoRule.class)
        public void incluirveiculo(){
                 conversation.begin();
                result.include("listaItensOpcionais", itemOpcionalDAO.findAllItensOpcionaisAtivos());

        }

        @Post
        @CustomBrutauthRules(LogadoRule.class)
        public void gravarveiculo(Veiculo veiculo, @Valid Marca marca, @Valid Modelo modelo, @Valid Cor
cor, CondicaoVeiculo condicaoVeiculo, UploadedFile[] files){

                this.atribuindoConfiguracaoDoVeiculo(veiculo);

                List<UploadedFile> listaImagens = Arrays.asList(files);
                veiculoValidator.validatorExisteImagemParaVeiculo(listaImagens);
```

```
		this.atribuiItensOpcionaisAoVeiculo(veiculo);
		this.atribuiImagensAoVeiculo(veiculo, files);
		this.atribuiCondicoesAoVeiculo(condicaoVeiculo);
		this.atribuiClintePropreitario(veiculo);

		if(marca.getId() != null){
			marca = marcaDAO.getById(marca.getId());
		}

		if(modelo.getId() != null){
			modelo = modeloDAO.getById(modelo.getId());
		}

		if(cor.getId() != null){
			cor = corDAO.getById(cor.getId());
		}

		veiculo.setMarca(marca);
		veiculo.setModelo(modelo);
		veiculo.setCor(cor);
		veiculo.setCondicaoVeiculo(condicaoVeiculo);
		condicaoVeiculo.setVeiculo(veiculo);

		validator.onErrorRedirectTo(this).incluirveiculo();

		veiculoValidator.validarVeiculo(veiculo);
		List<Message> errors = veiculoValidator.getValidator().getErrors();

		if(errors.isEmpty()){
			try {
				veiculoDAO.save(veiculo);
				conversation.end();
				result.forwardTo(this).meusveiculos();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}

		if(!errors.isEmpty()){
			veiculoValidator.getValidator().onErrorRedirectTo(this).incluirveiculo();
		}
	}

	private void atribuiClintePropreitario(Veiculo veiculo) {
		Cliente cliente = clienteDAO.getById(usuarioLogado.getCliente().getId());
		if(cliente.getVeiculos() != null){
			cliente.setVeiculos(new ArrayList<Veiculo>());
		}
		cliente.getVeiculos().add(veiculo);
		veiculo.setCliente(cliente);
	}

	private void atribuiCondicoesAoVeiculo(CondicaoVeiculo condicaoVeiculo) {
		if(condicaoVeiculo.getDataCadastro() == null){
			condicaoVeiculo.setDataCadastro(new Date());
		}
		if(condicaoVeiculo.getRemovido() == null){
			condicaoVeiculo.setRemovido(DOMINIO_SIM_NAO.NAO);
		}
		if(condicaoVeiculo.getSituacao() == null){
			condicaoVeiculo.setSituacao(DOMINIO_ATIVO_INATIVO.ATIVO);
		}
	}

	private void atribuiImagensAoVeiculo(Veiculo veiculo, UploadedFile[] files) {
		List<UploadedFile> listaImagens = Arrays.asList(files);
		for (UploadedFile uploadedFile : listaImagens) {
			Imagem imagem = new Imagem();
			imagem.abribuiImagem(uploadedFile);
			veiculo.getImagens().add(imagem);
		}
	}

	private void atribuindoConfiguracaoDoVeiculo(Veiculo veiculo) {

		try {
			String valorVeiculo = Arrays.asList(request.getParameterValues("valor")).get(0);
			DecimalFormat decFormat = new DecimalFormat("#,###.##");
			decFormat.setParseBigDecimal(true);
			veiculo.setPreco((BigDecimal) decFormat.parse( valorVeiculo.substring(3,
valorVeiculo.length()) ));
```

```
                } catch (Exception e) {

                }

                veiculo.setDataCadastro(new Date());
                veiculo.setPublicado(DOMINIO_SIM_NAO.SIM);
                veiculo.setCondicaoVeiculo(new CondicaoVeiculo());
                veiculo.setImagens(new ArrayList<Imagem>());
                veiculo.setItensOpcionais(new ArrayList<ItemOpcional>());
                veiculo.setFinalizado(DOMINIO_SIM_NAO.NAO);
                veiculo.setCertificado(DOMINIO_SIM_NAO.NAO);

                Calendar dataPublicacao = Calendar.getInstance();
                dataPublicacao.setTime(new Date());
                veiculo.setDataPublicacao(dataPublicacao);

        }

        private void atribuiItensOpcionaisAoVeiculo(Veiculo veiculo) {
                String[] itensOpcionaisSelecionados = request.getParameterValues("itemopcionais");
                try {
                        if(itensOpcionaisSelecionados != null){
                                if(itensOpcionaisSelecionados.length > 0){
                                        veiculo.setItensOpcionais(new ArrayList<ItemOpcional>());
                                        for(String itemOpcionalID : itensOpcionaisSelecionados){
                                                ItemOpcional itemOpcional =
itemOpcionalDAO.getById(Long.parseLong(itemOpcionalID));
                                                veiculo.getItensOpcionais().add(itemOpcional);
                                        }
                                }
                        }
                } catch (NumberFormatException e) {
                        veiculoValidator.getValidator().add(new I18nMessage("itemopcionais", "A lista de
itens opcionais pode esta com problema"));
                }
        }

}
```

Classe VeiculoValidator.java

```
package br.com.automag.validators;

import java.io.Serializable;
import java.util.List;

import javax.inject.Inject;

import br.com.automag.entity.veiculos.Veiculo;
import br.com.automag.persistencia.VeiculoDAO;
import br.com.caelum.vraptor.observer.upload.UploadedFile;
import br.com.caelum.vraptor.validator.Validator;
import br.com.paiter.core.factory.MessageFactory;

public class VeiculoValidator implements Serializable {


        private static final long serialVersionUID = 1L;

        // private static final int SIZE_MAX_FILE_ACCOUNT_BRONZE_6 = 6;
        private VeiculoDAO veiculoDAO;
        private Validator validator;
        private MessageFactory messageFactory;

        @Deprecated
        public VeiculoValidator() {}


        @Inject
        public VeiculoValidator(VeiculoDAO veiculoDAO, Validator validator, MessageFactory messageFactory)
{
                super();
                this.veiculoDAO = veiculoDAO;
                this.validator = validator;
                this.messageFactory = messageFactory;
        }
```

```
	public void validatorExisteImagemParaVeiculo(List<UploadedFile> listaImagens){
		if(listaImagens == null || listaImagens.isEmpty()){
			validator.add(messageFactory.build("error", "veiculo.errors.listaimagens.empty"));
		}
	}

	public Validator getValidator() {
		return validator;
	}


	public void validarVeiculo(Veiculo veiculo) {

		if(veiculo.getCliente() == null){
			validator.add(messageFactory.build("error", "veiculo.errors.cliente.empty"));
		}

		if(veiculo.getPreco() == null){
			validator.add(messageFactory.build("error", "veiculo.errors.preco.empty"));
		}

		if(veiculo.getTermosAceitos() == null){
			validator.add(messageFactory.build("error", "veiculo.errors.termoaceito.empty"));
		}

	}

}
```

Camada de Visualização.

Index.jsp

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>

<fmt:message key="site.name" var="siteName" />

<tags:automag-start-page-content />

	<tags:header facebookMetas="${true}" title="${genericTitle} - ${title}" description="$
{description}"/>

<div id="page-content">
	<section id="car-advertisement">
		<div class="content-holder">
			<div class="slide-part-large">
				<div class="visible-image">
					<h2><span class="bold">Ofertas</span> em destaque</h2>
					<div id="slides">
						<ul class="slides">
							<c:forEach items="${veiculosEmDestaque}"
var="veiculo">
								<li>
<a href="$
{linkTo[VeiculoController].exibirVeiculo(veiculo.id, veiculo.modelo.nome)}">
<img src="$
{linkTo[ImagemController].imagemUnicaDoVeiculo(veiculo.id)}" alt="Slide" />
<div class="slide-info">
<h3>$
{veiculo.modelo.nome}</h3>
<span
class="price-tag">${veiculo.preco}</span>
</div>
</a>
</li>
</c:forEach>
</ul>
</div>
</div>
</div>

<div class="latest-offers">
<div class="headline">
<h2><span class="bold">Últimas</span> ofertas</h2>
<a href="#" class="scroll-up scroll-icon"><span>Up</span></a>
```

```
                                        <a href="#" class="scroll-down scroll-icon"><span>Down</span></a>
                                </div>

                                <ul class="offer-list offer-small list-content">
                                        <c:forEach items="${ultimosVeiculos}" var="veiculoUltimo">
                                                <li>
                                                        <a href="car-details.html">
                                                                <img src="$
{linkTo[ImagemController].imagemUnicaDoVeiculo(veiculoUltimo.id)}" alt="Acesse essa oferta" />
                                                                <div class="entry-label">
                                                                        <h4>$
{veiculoUltimos.modelo.nome}</h4>
                                                                        <span class="price-tag">$
{veiculoUltimo.preco} Reais</span>
                                                                </div>

                                                                <div class="entry-overlay">
                                                                        <ul class="car-list-details">
                                                                                <li>$
{veiculoUltimo.anoFabricacao}</li>
                                                                                <li>$
{veiculoUltimo.tipoCombustivel.desc}</li>
                                                                                <li>$
{veiculoUltimo.motorizacao.desc}</li>
                                                                                <li>$
{veiculoUltimo.tipoVeiculo.desc}</li>
                                                                                <li>$
{veiculoUltimo.quilometragem} KM</li>
                                                                        </ul>
                                                                </div>
                                                                <span class="v-sign">V</span>
                                                                <span class='dealer-data'>Detalhe do
veículo</span>
                                                        </a>
                                                </li><!--/end of item 1-->
                                        </c:forEach>
                                </ul>


                        </div><!--.latest-offers-->
                </div>

        </section><!--#car-advertisement-->

        <section id="car-shortcuts">
                <div class="content-holder">

                        <div class="full-width banners-full-width">
                                <div class="search-banner banner-medium one-half">
                                        <a href="car-list.html" class="icon-magnify icon-
banner">Magnify</a>
                                        <div class="banner-title">
                                                <h3><span class="bold">Looking</span> for a car?</h3>
                                                <p class="description">1.000 new offers every day. 35.000
offers on site</p>
                                        </div>
                                        <a href="car-list.html" class="banner-
link"><span>Search</span></a>
                                </div>

                                <div class="sell-banner banner-medium one-half">
                                        <a href="add-vehicle.html" class="icon-dollar icon-
banner">Dollar</a>
                                        <div class="banner-title">
                                                <h3><span class="bold">Want</span> to sell a car?</h3>
                                                <p class="description">200.000 visitors every day. Add
your offer now!</p>
                                        </div>
                                        <a href="add-vehicle.html" class="banner-
link"><span>Sell+</span></a>
                                </div>
                        </div>

                        <div class="full-width articles-dealers-offers">
                                <div class="popular-articles one-third">
                                        <h3><span class="bold">Popular</span> Articles</h3>
                                        <ul class="articles-list">
                                                <li>
                                                        <img src="images/article_icon.gif" alt="Article
Icon" />
```

```
                                                <p class="preview">Here are many variations of
passages of Lorem Ipsum..</p>
                                                <a href="blog-post.html" class="more-link">Read
more</a>
                                        </li>
                                        <li>
                                                <img src="images/article_icon.gif" alt="Article
Icon" />
                                                <p class="preview">Here are many variations of
passages of Lorem Ipsum..</p>
                                                <a href="blog-post.html" class="more-link">Read
more</a>
                                        </li>
                                </ul>
                        </div><!--.popular-articles-->

                        <div class="car-dealers one-third">
                                <div class="headline">
                                        <h3><span class="bold">Car</span> dealers</h3>
                                        <a href="dealer-list.html" class="see-more">See all
dealers</a>
                                </div>

                                <ul class="dealer-list">
                                        <li>
                                                <a href="dealer-details.html" class="dealer-
name">Dealer name</a>
                                                <span class="amount-offers">162 Offers</span>
                                        </li>
                                        <li>
                                                <a href="dealer-details.html" class="dealer-
name">Dealer name</a>
                                                <span class="amount-offers">162 Offers</span>
                                        </li>
                                        <li>
                                                <a href="dealer-details.html" class="dealer-
name">Dealer name</a>
                                                <span class="amount-offers">162 Offers</span>
                                        </li>
                                        <li>
                                                <a href="dealer-details.html" class="dealer-
name">Dealer name</a>
                                                <span class="amount-offers">162 Offers</span>
                                        </li>
                                </ul>
                        </div><!--.car-dealers-->

                        <div class="daily-offers one-third">
                                <h3><span class="bold">Get</span> daily offers</h3>
                                <form action="#" method="post">
                        <input type="text" onfocus="if(this.value == 'E-mail here') { this.value = ''; }"
onblur="if(this.value == '') { this.value = 'E-mail here'; }" value="E-mail here" class="email-address
default-input ">
                                        <p class="offer-text">Available, but have suffered
alteration in some form injected humour, or randomised words which don't look even slightly believable.
</p>
                                        <div class="submit-button"><input type="submit"
value="Subscribe" /></div>
                                        <p class="amount-subscribers">15,000+ other
subscribers</p>
                                </form>
                        </div><!--.daily-offers-->
                </div>
        </div>
</section><!--#car-shortcuts-->

</div><!--#page-content-->

<tags:automag-end-page-content/>
```