UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

RELATÓRIO DE ESTÁGIO

DESENVOLVIMENTO DO MACROMÓDULO DE CONTROLE E MANUTENÇÃO DE VEÍCULOS DO MINISTÉRIO PÚBLICO DA PARAÍBA

ANTONIO ALEXANDRE MOURA COSTA

Estagiário

JOSÉ EUSTÁQUIO RANGEL DE QUEIROZ

Orientador Acadêmico

UIRÁ VASCONSELOS ALENCAR DE ASSIS

Supervisor Técnico

Campina Grande – PB

10 de dezembro de 2010

DESENVOLVIMENTO DO MACROMÓDULO DE CONTROLE E MANUTENÇÃO DE VEÍCULOS DO MINISTÉRIO PÚBLICO DA PARAÍBA

APROVADO EM _____

BANCA EXAMINADORA

_____

Prof. José Eustáquio Rangel de Queiroz, DSc.

ORIENTADOR ACADÊMICO

_____

Prof.ª Joseana Macêdo Fechine, DSc.

MEMBRO DA BANCA

_____

Prof. Leandro Marinho, DSc.

MEMBRO DA BANCA

Biblioteca Setorial do CDSA. Maio de 2021.

Sumé - PB

AGRADECIMENTOS

Agradeço ao Ministério Público da Paraíba (MPPB), pela oportunidade de participação no estágio e assim conseguir subir mais um degrau na minha vida acadêmica e profissional.

Agradeço ao meu orientador acadêmico Prof. José Eustáquio Rangel de Queiroz, pela paciência, disponibilidade e apoio que sempre forneceu, durante esse período.

Ao chefe do departamento de desenvolvimento do MPPB, que guiou o desenvolvimento do problema abordado no estágio.

A minha família, que sempre forneceu todo o apoio necessário para seguir buscando novas maneiras de crescer pessoal e profissionalmente.

A Mithylenny, pela compreensão, paciência, amor, carinho e total apoio que sempre demonstrou, durante todo o tempo do período de estágio.

Aos colegas de estágio que constituíram uma grande equipe de desenvolvimento e suporte técnico, auxiliando na resolução dos problemas diários enfrentados.

APRESENTAÇÃO

Como parte das exigências do curso de Ciência da Computação, da Universidade Federal de Campina Grande, para cumprimento da disciplina de estágio integrado, apresenta-se o relatório de estágio, DESENVOLVIMENTO DO MACROMÓDULO DE CONTROLE E MANUTENÇÃO DE VEÍCULOS DO MINISTÉRIO PÚBLICO DA PARAÍBA.

O estágio foi realizado no 2º Centro de Apoio Operacional às Curadorias, situado à Rua Promotora Terezinha Lopes de Moura S/N, Liberdade, 58.410-064, Campina Grande Paraíba.

O conteúdo do relatório está distribuído conforme segue:

Seção 1 – Introdução: aborda a importância do estágio na formação do aluno, a delimitação e os objetivos do estágio.

Seção 2 – Ambiente de estágio: aborda o local de trabalho, identificando a empresa no qual o estágio foi realizado, breve descrição das atividades desenvolvidas e a infraestrutura do estágio.

Seção 3 – Fundamentação teórica: aborda a fundamentação teórica, trazendo informações sobre as tecnologias e aplicativos utilizados.

Seção 4 – Atividades do estágio: aborda detalhadamente as tarefas realizadas durante o tempo de estágio.

Seção 5 – Considerações finais.

Referências bibliográficas.

O Apêndice A contém o plano de estágio proposto inicialmente, enquanto os Anexos A, B e C contêm os códigos-fonte relacionados a *Cadastro de Locadores*.

RESUMO

O presente relatório documenta as atividades realizadas no estágio integrado no período de 8 de setembro de 2010 a 2 de dezembro de 2010. Ao longo dos últimos meses foram realizadas atividades bastante diversificadas no MPPB, desde tarefas ligadas ao suporte técnico até o desenvolvimento do macro módulo de Controle e Manutenção de Veículos. Na área de suporte, onde as tarefas surgiam com mais freqüência, tinha-se basicamente as seguintes atividades: detecção e resolução de problemas ligados tanto a *hardware* como *software*, manutenção da rede inter do MPPB e suporte ao usuário no uso dos aplicativos adotados pelo órgão. No tocante as tarefas de desenvolvimento foram implementados os submódulos Cadastro de Locadores, que consiste no registro das locações de veículos realizadas pelos servidores do MP e *Cadastro de Condutores*, e *Cadastro de Condutores* que visa catalogar os servidores que desempenham função de condutor para o órgão.

SUMÁRIO

LISTA DE SIGLAS E ABREVIATURAS

AMD	*Advanced Micro Devices*
AOP	*Aspect Oriented Programming*
API	*Application Programming Interface*
BSD	*Berkeley Software Distribution*
CD	*Compact Disc*
DAO	*Data Access Object*
DVD	*Digital Video Disc*
EJB	*Enterpise JavaBeans*
GC	Gerenciamento de Controle
GUI	*Graphical User Interface*
HD	*Hard Disk*
HTML	*HyperText Markup Language*
HTTP	*HypertText Tranfer Protocol*
IDE	*Integrated Development Environment*
IoC	*Inversion of Control*
ISO	*International Organization for Standardization*
JDBC	*Java DataBase Conectivity*
JDT	*Java Development Tooling*
JNDI	*Java Naming and Diretory Interface*
JSF	*Java Server Faces*
JSP	*Java Server Pages*
JVM	*Java Virtual Machine*
JTA	*Java Transaction API*
KDE	*K Desktop Environment*
MP	Ministério Público
MPPB	Ministério Público da Paraíba
MVC	*Model View Controller*
MVCC	*MultiVersion Concurrency Control*
ORM	*Object Relational Mapping*
PC	*Personal Computer*
POO	Programação Orientada a Objeto
RAM	*Random Access Memory*
SGBD	Sistema de Gerenciamento de Banco de Dados
SVN	*SubVersioN*
SQL	*Structure Query Language*
UI	*User Interface*

XML	*eXtensible Markup Language*
XP	*eXtreme Programming*

LISTA DE FIGURAS

LISTA DE QUADROS

SEÇÃO I

INTRODUÇÃO

1 Introdução

A participação em um estágio mostra-se de fundamental importância, pois é mediante tal elemento acadêmico que se tem a oportunidade de aplicar e melhorar todo o conhecimento adquirido durante a vida acadêmica, e desta forma conseguir adentrar no mercado de trabalho e ter seu reconhecimento profissional mais facilmente.

1.1 Delimitação do estágio

O estágio foi realizado no setor de informática do MPPB, na cidade de Campina Grande, tendo início no mês de setembro de 2010 e conclusão no mês de dezembro de 2010, período correspondente ao semestre letivo de 2010.2 da Universidade Federal de Campina Grande.

1.2 Objetivos

A seguir, são descritos os objetivos, geral e específicos, identificados no estágio integrado.

1.2.1 Objetivo geral

Desenvolver o macromódulo* de Controle e Manutenção de Veículos, que faz parte do *Sistema Aristóteles*, que foi criado pelo MPPB.

1.2.2 Objetivos específicos

O objetivo geral é dividido de seguinte forma:

- Implementar o submódulo* *Cadastro de Locadores*, que é responsável pelo registro de todos aqueles que necessitam realizar a locação de qualquer veículo do MPPB.
- Implementar o submódulo *Cadastro de Condutores*, que é responsável pelo registro de todos os condutores de veículos do MPPB.
- Detectar e resolver todos os problemas relacionados a *hardware* e *software* e realizar freqüentemente a manutenção da rede interna do órgão.

** Termos definidos no plano de estágio (Apêndice A) pelo Departamento de desenvolvimento do MPPB.*

SEÇÃO II

AMBIENTE DE ESTÁGIO

2 Ambiente de estágio

O ambiente de estágio no que diz respeito à estrutura física se mostrou bastante precário. Infelizmente, o *hardware* disponível era antigo e lento, além do que, se qualquer reposição de peças que fosse necessária se tornava inviável devido a grande burocracia, que é característica marcante dos órgãos públicos.

2.1 Identificação da empresa

O MP atua no campo dos interesses daqueles que por si só são insuficientes para se defender e vela pela aplicação da lei.

O Ministério Público é instituição permanente, essencial à função jurisdicional do Estado, incumbindo-lhe a defesa da ordem jurídica, do regime democrático e dos interesses sociais e individuais indisponíveis.

Na Figura 01, observa-se como é dividida a estrutura administrativa do órgão na área de Tecnologia da Informação.

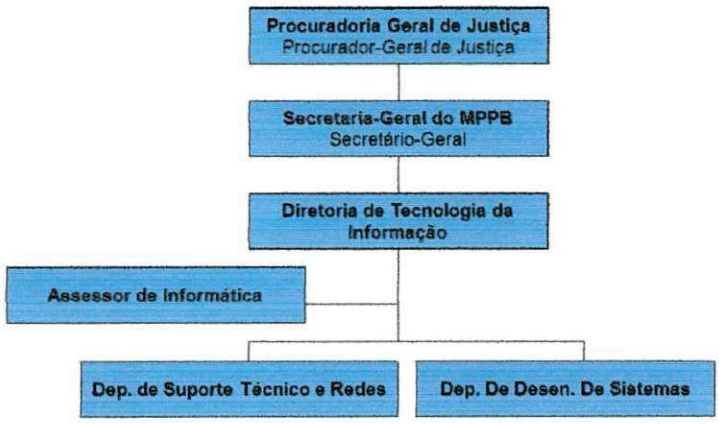

Figura 01 - Organograma Administrativo do MPPB.

2.2 Infraestrutura de *hardware* e *software*

O MPPB dispõe de um Setor de Informática que conta com três computadores. No entanto, apenas um deles se adequava ao desenvolvimento. Diante desta situação, foi necessário utilizar uma

máquina pessoal, que contava com a seguinte configuração:

- Processador Core i5 450M 2,4 GHz;
- 4 GB de memória RAM DDR3 1066 MHz;
- 500 GB de espaço em disco;
- Sistemas Operacionais Windows 7 e Ubuntu 10.02, ambos 64 bits.

Um DVD, com uma distribuição Linux e vários aplicativos que seriam necessários para o desenvolvimento, foi fornecido pelo chefe do departamento de desenvolvimento, para dar início às atividades relacionadas ao macromódulo em questão.

SEÇÃO III

FUNDAMENTAÇÃO TEÓRICA

3 Fundamentação teórica

Nesta seção, são descritas com detalhes, as tecnologias e aplicativos utilizados durante o desenvolvimento do sistema em questão. Tais tecnologias já eram usadas pelo Sistemas Aristóteles, assim seu aprendizado era necessário para dar início a implementação.

3.1 Linguagem de programação: *Java*

Tendo sido originalmente concebida para o desenvolvimento de pequenos aplicativos e programas de controle de aparelhos eletrodomésticos e eletroeletrônicos, a linguagem de programação *Java* mostrou-se ideal para ser usada na internet. O que a torna tão atraente é o fato de programas escritos em *Java* poderem ser executados virtualmente em qualquer plataforma, mas principalmente em *Windows, Unix e Mac*. Soma-se a isto o fato de programas *Java* poderem ser embutidos em documentos *HTML* (*HiperText Markup Language*), podendo assim ser divulgados pela rede. Diferente da linguagem C, não é apenas o código fonte que pode ser compartilhado pela rede, mas o próprio código executável compilado, chamado *bytecode* (LARA, 2010).

Java foi desenvolvida por um grupo de pesquisadores da *SUN Microsystems* nos anos 90, pouco antes da explosão da Internet. Essa linguagem possui estrutura muito semelhante à da linguagem C, da qual descende imediatamente. O *Java* tem em comum com a linguagem C++ o fato de ser orientada a objetos e mantém com esta um alto grau de semelhança. O paradigma de programação orientada a objetos consiste de um grau a mais na abstração da programação, em comparação com a programação estruturada, e tem se mostrado extremamente útil na produção de programas cada vez mais sofisticados, em menor tempo e com maior qualidade. A programação orientada a objetos (POO) é hoje universalmente adotada como padrão de mercado.

Há certa curiosidade por trás do nome dado a esta linguagem de programação. *Java* é o nome de uma ilha do Pacífico, onde se produz certa variedade de café homônimo. A inspiração bateu à equipe de desenvolvimento ao saborear esse café em uma lanchonete local. Deram-se conta de como era extremamente apreciado por profissionais da área de *software* (ao menos nos Estados Unidos), de modo que não foi menos justo fazer-lhe homenagem ao batizar uma nova linguagem de programação.

3.2 *JavaServer Faces (JSF)*

JSF é uma tecnologia que incorpora características de um *framework MVC (Model View Controller)* para *Web* e de um modelo de interfaces gráficas baseado em eventos. Por basear-se no padrão de projeto *MVC*, uma de suas melhores vantagens é a clara separação entre a visualização e regras de negócio (modelo).

A idéia do padrão *MVC* é dividir uma aplicação em três camadas: modelo, visualização e controle. O modelo é responsável por representar os objetos de negócio, manter o estado da aplicação e fornecer ao controlador o acesso aos dados. A visualização representa a interface com o usuário, sendo responsável por definir a forma como os dados serão apresentados e encaminhar as ações dos usuários para o controlador. A camada de controle, por sua vez é responsável por fazer a ligação entre o modelo e a visualização, além de interpretar as ações do usuário e as traduzir para uma operação sobre o modelo, onde são realizadas mudanças e, então, gerar uma visualização apropriada.

No *JSF*, o controle é composto por um *servlet* denominado *FacesServlet*, por arquivos de configuração e por um conjunto de manipuladores de ações e observadores de eventos. O *FacesServlet* é responsável por receber requisições da *Web*, redirecioná-las para o modelo e então remeter uma resposta. Os arquivos de configuração

são responsáveis por realizar associações e mapeamentos de ações e pela definição de regras de navegação. Os manipuladores de eventos são responsáveis por receber os dados vindos da camada de visualização, acessar o modelo, e então devolver o resultado para o *FacesServlet* (PITANGA, 2010).

O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização. Finalmente, a visualização é composta por *component trees* (hierarquia de componentes *UI*), tornando possível unir um componente ao outro para formar interfaces mais complexas. Na Figura 02, mostra-se a arquitetura do *JavaServer Faces* baseada no modelo *MVC*.

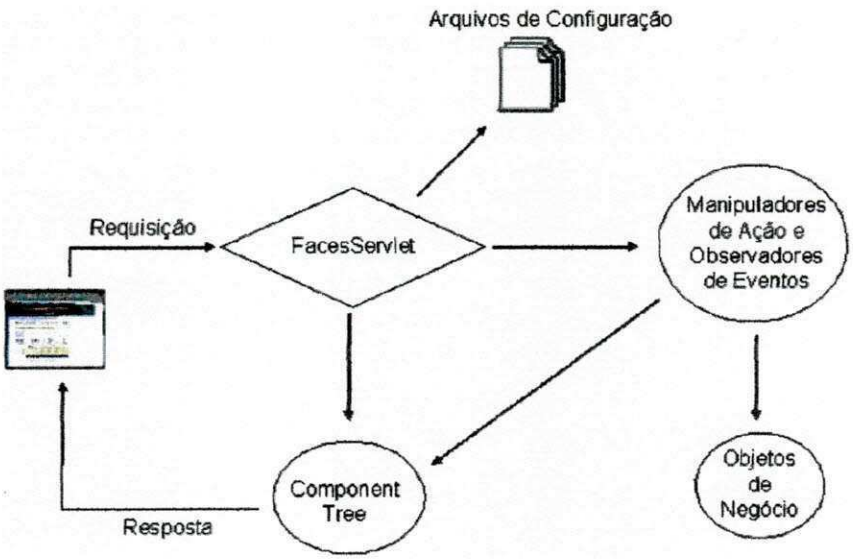

Figura 02 - Arquitetura *JSF* baseada no modelo *MVC*.

Embora *JavaServer Faces* forneça *tags JSP* (*Java Server Pages*) para representar os componentes em uma página, ele foi projetado para ser flexível, sem limitar-se a nenhuma linguagem em particular, nem a protocolos ou tipo de clientes. Ele também permite a criação de componentes próprios a partir de classes de componentes, conforme mencionado anteriormente.

JSF possui dois principais componentes: *Java API* para a

representação de componentes *UI* e o gerenciamento de seus estados, manipulação/observação de eventos, validação de entrada, conversão de dados, internacionalização e acessibilidade; e *taglibs JSP* que expressam a interface *JSF* em uma página *JSP* e que realizam a conexão dos objetos no lado servidor.

3.3 Hibernate

O *Hibernate* é um *framework* para o mapeamento objeto-relacional escrito na linguagem *Java*, mas também está disponível para a plataforma *.Net* com o nome de *NHibernate*. Ele facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (*XML*) para estabelecer esta relação.

Seu objetivo é diminuir a complexidade entre os programas *Java*, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacional, presente na maioria dos Sistemas de Gerenciamento de Banco de Dados (SGDBs). Em especial, no desenvolvimento de consultas e atualizações dos dados.

Tem como principal característica a transformação das classes em *Java* para tabelas de dados (e dos tipos de dados *Java* para os da *SQL*). O *Hibernate* gera as chamadas *SQL* e libera o desenvolvedor do trabalho manual da conversão dos dados resultante, mantendo o programa portável para quaisquer bancos de dados *SQL*, porém causando um pequeno aumento no tempo de execução.

As questões relacionadas para o gerenciamento de transações e na tecnologia de acesso à base de dados são de responsabilidade de outros elementos na infraestrutura do programa. Apesar de existirem *API* no *Hibernate* para possuir operações de controle transacional, ele simplesmente delegará estas funções para a infraestrutura na qual foi instalada (OLIVEIRA, 2010).

No caso de aplicações construídas para serem executadas em servidores de aplicação, o gerenciamento das transações é realizado segundo o padrão *JTA (Java Transaction API)*. Nas aplicações *standalone*, o programa delega o tratamento transacional ao *driver* JDBC (*Java DataBase Conectivity*).

O *Hibernate* foi criado por desenvolvedores *Java*, espalhados ao redor do mundo, e liderado por Gavin King. Posteriormente, a *JBoss Inc* (empresa comprada pela *Red Hat*) contratou os principais desenvolvedores do programa para fazer o seu suporte. Ele pode ser utilizado em aplicações *Java standalone* ou em aplicações *Java EE* (*Enterprise Edition*), utilizando *servlet* ou sessões *EJB (Enterprise JavaBeans)*.

3.4 Spring

O *Spring* é um *framework open source* criado, não intrusivo, baseado nos padrões de projeto inversão de controle (*IoC*) e injeção de dependência.

A inversão de controle é o que torna uma simples biblioteca de classes diferente de um *framework*. Uma biblioteca consiste em um conjunto de classes que um usuário instancia e utiliza seus métodos. Após a chamada ao método, o controle do fluxo da aplicação retorna para o usuário. Entretanto, em um *framework* este fluxo é diferente. Para utilizar um *framework*, código próprio da aplicação deve ser criado e mantido acessível ao mesmo, podendo ser a partir de classes que estendem classes do próprio. Ele, então, realiza a chamada deste código da aplicação. Após a utilização do código da aplicação, o fluxo retorna para ele.

O padrão injeção de dependência consiste em uma especialização do padrão inversão de controle. Aplicações como Spring e *PicoContainer*, denominados de *lightweight*

containers, adotam a inversão de controle (CARVALHO, 2010).

Na Figura 03, é apresentada a estrutura do *Spring*:

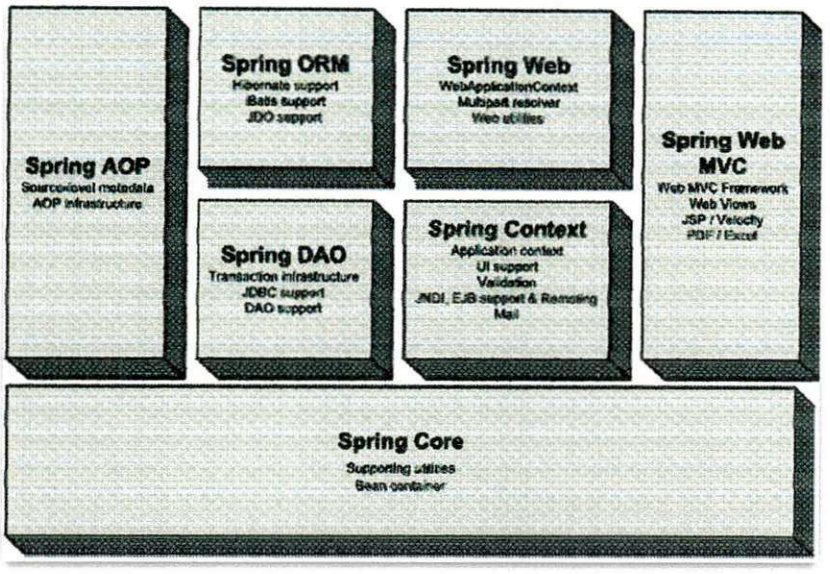

Figura 03 – Estrutura do Spring.

O módulo *Spring Core* representa as principais funcionalidades do *Spring*, no qual o principal elemento é o *BeanFactory*. Trata-se de uma implementação do padrão *Factory*, responsável em remover a programação de *Singletons* e permitindo o baixo acoplamento entre a configuração e a especificação de dependências, de sua lógica de programação.

O módulo *Spring DAO* provê uma camada de abstração para *JDBC*, eliminando grande parte da codificação necessária para interagir com um banco de dados. O módulo *ORM*, entretanto, provê integração do *Spring* com outros frameworks para persistência de objetos, como *Hibernate* e *iBatis*. Para prover uma implementação orientada a aspectos que permite a definição de *pointcuts* e *methods interceptors*, existe o módulo *Spring AOP*.

Para prover funcionalidades específicas para projetos Web, tem-se o módulo *Spring Web*. São funcionalidades como componentes para *upload* de arquivos e suporte para utilização de

Inversão de Controle neste tipo de aplicação. O módulo *Spring MVC*, entretanto, fornece uma implementação de *framework* Web, similar ao *Struts*.

Inicialmente, é preciso entender o conceito, adotado pelo *Spring*, de *beans*. Para este framework, qualquer objeto que forma sua aplicação e que está sob controle do *Spring*, é considerado um bean. Enfim, um *bean* trata-se apenas de um objeto de sua aplicação e nada mais. O *Container IoC* é o responsável pelo gerenciamento destes *beans*.

Estes *beans*, entretanto, muitíssimo provavelmente possuem dependências entre si. Estas dependências são definidas a partir de metadados. O Container *IoC* obtém essas configurações e, partindo destas configurações, gerencia a dependência entre os *beans*. Neste contexto, a interface *BeanFactory* representa o Container *IoC* do *Spring*. Existem diversas implementações de *BeanFactory*, sendo a *XmlBeanFactory* a implementação mais comum. Nesta, toda configuração de dependência entre os objetos é definida em um arquivo *XML*.

3.5 Aplicativos utilizados

A utilizados dos *softwares* descritos nesta seção foi estabelecida pelo chefe do departamento de desenvolvimento do MPPB.

3.5.1 Sistema operacional: Ubuntu 10.02 (64 bits)

Ubuntu é uma antiga palavra Africana, cujo significado é "humanidade para todos". *Ubuntu* também quer dizer "E sou o que sou devido ao que todos nós somos". A distribuição *Ubuntu Linux* traz o espírito do *Ubuntu* ao mundo do *software*. É baseada em *Debian*, com edições regulares (a cada seis meses), com uma grande atenção nos utilizadores e na facilidade de uso e um comprometimento com suporte e atualizações de segurança durante 18 meses para cada

edição. O *Ubuntu* usa as versões mais recentes do *Gnome* e uma seleção de *software* (para desktop e servidores) num simples *CD* (ou *DVD*) de instalação. Estão também disponíveis outros ambientes de trabalho, em geral suportados pela comunidade como é o caso do *KDE*. Existem também *CD/DVD* de instalação focados para um ou outro segmento de utilizadores, como é o caso do *Edubuntu* (Versão uso em escolas). Com uma instalação base *Ubuntu* pode facilmente converter entre diferentes ambientes de trabalho ou aplicações instaladas por omissão.

A proposta do *Ubuntu* é oferecer um sistema operativo que qualquer pessoa possa utilizar sem dificuldades, independentemente de nacionalidade, nível de conhecimento ou limitações físicas. A distribuição deve ser constituída totalmente de *software* gratuito e livre, além de isenta de qualquer taxa. Atualmente, uma organização cuida para que cópias sejam remetidas em *CD* para todo o mundo sem custos.

Focando-se na qualidade, o *Ubuntu* produz um ambiente de computação rico e robusto que é flexível para usar tanto em casa como em ambientes empresariais. O projeto tem mais tempo para despender em detalhes e é capaz de lançar uma versão que inclui as melhores e mais atuais aplicações gratuitas conhecidas a cada seis meses.

3.5.2 Apache Tomcat 7.0.4

O *Apache Tomcat* é um servidor de aplicações *Java* para Web que implementa as tecnologias *JavaServlets* e *JavaServer Pages*. Ele também pode comportar-se como um servidor Web (*HTTP*) ou funcionar integrado a um servidor Web dedicado (como o *Apache* ou o *IIS*).

Este programa é um *software* livre (e uma aplicação de código aberto), nascido no Projeto *Apache* Jakarta e é oficialmente

autorizado pela *Sun* (desenvolvedora do Java) como a implementação de referência para as tecnologias *Java Servlet* e *JavaServer Pages*. Ele cobre parte da especificação *J2EE* com tecnologias como *servlet* e *JSP*, e tecnologias de apoio relacionadas como *Realms* e segurança, *JNDI Resources* e *JDBC DataSources*, contudo, ele não implementa pacotes *EJB* (*Enterprise JavaBeans*).

O *Apache Tomcat* é inteiramente escrito em *Java* e, portanto, para ser executado em seu computador ele necessita de uma *Java Virtual Machine* (*JVM*) instalada. A instalação do servidor é simples, porém a configuração requer conhecimento prévio sobre o assunto ou uma leitura criteriosa nos manuais encontrados tanto na página do desenvolvedor como em vários sites na Internet.

3.5.3 IDE: *Eclipse* Helios

O *Eclipse* é um *framework* para integrar diferentes tipos de aplicações. Uma de suas aplicações é a *JDT* (*Java Development Tooling*), a qual já vem com o Eclipse.

Essas aplicações são oferecidas em forma de *plugins* e automaticamente reconhecidas e integradas pela plataforma. Tendo seus próprios recursos para gerenciamento de mecanismo, que são geralmente arquivos no *HD*. Eles residem no seu *workspace*, uma pasta especial localizada no sistema de arquivos. As aplicações instaladas comunicam-se entre si, com isso, se uma aplicação altera um recurso qualquer, todas as outras aplicações instaladas serão notificadas sobre essa mudança, garantindo uma consistência e integração em todo o seu ambiente de desenvolvimento.

Um usuário sempre trabalha no *workbench*, a parte "visível" da plataforma (*GUI*). A perspectiva escolhida determina a aparência atual do *workbench*. A perspectiva é uma coleção conhecida como "*views* e editores" que contribuem com ações especiais para o menu e a toolbar. A maioria das *views* mostra informações especiais sobre

os recursos. Dependendo da *view* somente partes ou relacionamentos internos dos recursos poderão ser mostrados. Como no caso de arquivos do tipo *XML* (VALENTIM, 2010).

Um editor trabalha diretamente sobre um recurso (classes Java como exemplo). O Eclipse segue um ciclo de carrega-altera-salva que somente se um editor salvar suas alterações, a plataforma será capaz de notificar as outras aplicações. *Views* especiais podem ser conectadas diretamente a um editor, adicionando recursos complementares ao manuseamento dos recursos. Por exemplo, a *Outline View* da perspectiva *Java* é conectada diretamente ao Editor *Java*.

O que torna o *Eclipse* um IDE especial, é a extrema flexibilidade na qual podem ser combinadas *views* e editores. Dessa forma o *workbench* pode ser arrumado de uma forma livre e que melhor adapte o desenvolvedor. As *views* e editores podem ser adicionados em uma perspectiva aberta (mesmo se eles foram definidos em um *plugin* totalmente diferente). Portanto, é possível ter a total liberdade para criar o ambiente de desenvolvimento que melhor agrade ao desenvolvedor, de uma forma agradável e customizada.

3.5.4 PostgreSQL 8.4.0

O banco de dados *PostgreSQL* nasceu na Universidade de *Berkeley*, nos anos 80, como um projeto acadêmico, sendo mantido pela comunidade *Open Source*. A coordenação do desenvolvimento do *PostgreSQL* é executada pelo *Global Development Group*, este é formado por um amplo grupo de desenvolvedores ao redor do mundo, o que faz o *PostgreSQL* ter uma constante evolução no que se refere à correção de *bugs* e implementação de novas funcionalidades (SILVA, 2010).

A primeira versão funcional (versão 1) foi liberada para um

grupo pequeno de usuários em junho de 1989. Após a liberação da versão 4 (1993), quando o sistema já contava com uma boa popularidade, o projeto foi oficialmente abandonado pela Universidade de Berkeley e sua continuação só foi possível porque o mesmo estava sob uma licença *BSD*. Em 1994, dois estudantes de Berkeley, Andrew Yu e Jolly Chen, adicionaram um interpretador *SQL* ao *Postgres*, deram ao projeto o nome de Postgres95 e divulgaram seu código pela Internet. Com o código aberto (open source), o Postgres95 extrapolou as fronteiras da universidade permitindo que outros desenvolvedores se integrassem ao projeto. Logo, em 1996, o projeto foi renomeado para *PostgreSQL*, que refletia melhor a nova linguagem de consulta ao banco de dados, o *SQL*. Desde então, o *PostgreSQL* não parou de crescer, sendo mantido por um grupo de desenvolvedores e de voluntários de todo o mundo. A partir da versão 6.0, o agora *PostgreSQL*, passou a suportar o *MVCC* (*Multiversion Concurrency Control*).

Outro marco na história do *PostgreSQL* aconteceu com o lançamento em janeiro de 2005 da versão 8.0, que trouxe, entre outras novidades, o suporte nativo para Microsoft Windows (até então eram utilizados emuladores para executar o *PostgreSQL* no Windows).

O *PostgreSQL* é um SGBDRe, ou seja, é um Sistema de Gerência de Bancos de dados Relacional estendido e livre. "R" porque ele adota uma abordagem relacional. A abordagem em um SGBD é caracterizada pela forma como os dados estão organizados nesse SGBD. Em um SGBD relacional, os dados estão organizados em forma de tabelas (linhas e colunas) e suas relações (chaves estrangeiras). Notem que essa é uma definição bastante simplista de um banco de dados relacional. O "e" em SGBDRe é de estendido, e denota que o PostgreSQL não está restrito à abordagem relacional. Futuramente veremos algumas dessas funcionalidades.

A utilização da dupla *PostgreSQL* e *Linux* nas empresas vem crescendo rapidamente. Também é possível utilizar o *PostgreSQL* no ambiente *MS Windows*, mas como ele não foi portado nativamente para este sistema operacional, seu desempenho no *Linux* é sensivelmente maior e mais estável do que no *Windows*.

Tecnicamente, é possível utilizá-lo em qualquer projeto que necessite de um banco de dados. Entretanto, para empresas que ainda não possuem uma cultura *Linux* ou *PostgreSQL*, é recomendado começar por sistemas periféricos e partir gradativamente para os sistemas críticos.

Uma das características mais destacadas do *PostgreSQL* é a sua capacidade de lidar com um grande volume de dados. Existem aplicações em produção com tabelas possuindo mais de 100 milhões de linhas. Nos testes de carga realizados pela *Dextra* Sistemas, foram feitas simulações com tabelas com 200 milhões de linhas e os resultados obtidos foram excelentes. No Brasil, existem casos de sucesso interessantes de algumas empresas de *telecom* lidando com base de dados com dezenas de milhões de registros gerenciadas pelo *PostgreSQL* em servidores *Linux* e arquitetura padrão *Intel*.

SEÇÃO IV

ATIVIDADES REALIZADAS

4 Atividades realizadas

O cronograma proposto no Plano de estágio (Apêndice A), não foi seguido completamente. Foram feitas alterações em relação às atividades específicas de cada membro da equipe e a metodologia de desenvolvimento.

Atividade 1: Levantamento de requisitos que envolviam estudo de viabilidade, elicitação, especificação e validação de requisitos junto ao chefe de departamento de desenvolvimento. Realizada entre 13 de setembro de 2010 e 24 de setembro de 2010, totalizando 30 horas.

Atividade 2: Estudo das tecnologias JSF, Hibernate, Spring, Servlets que seriam utilizadas no desenvolvimento dos módulos. Realizada entre 27 de setembro de 2010 e 08 de outubro de 2010, totalizando 30 horas.

Atividade 3: Configuração do ambiente *Linux*, do Eclipse, *TomCat* e do controle de versão. Realizada entre 11 de outubro de 2010 e 19 de outubro de 2010, totalizando 25 horas.

Atividade 4: Desenvolvimento de uma entidade para familiarização com a estrutura do código. Realizada entre 22 de setembro de 2010 e 29 de outubro de 2010, totalizando 35 horas.

Atividade 5: Desenvolvimento do submódulo *Cadastro de Locadores*. Realizada entre 01 de novembro de 2010 e 15 de novembro de 2010, totalizando 55 horas.

Atividade 6: Desenvolvimento do submódulo *Cadastro de Condutores*. Realizada entre 16 de novembro de 2010 e 15 de novembro de 2010, totalizando 25 horas.

Atividade 7: Integração com o submódulo *Cadastro de Veículos*. Realizada entre 16 de novembro de 2010 e 23 de novembro de 2010, totalizando 15 horas.

Atividade 8: Integração com o submódulo *Cadastro de Locadores*. Realizada entre 24 de novembro de 2010 e 26 de novembro de 2010, totalizando 10 horas.

Atividade 9: Suporte técnico. Realizada entre 01 de novembro de 2010 e 15 de novembro de 2010, totalizando 70 horas.

Inicialmente, estava previsto o desenvolvimento dos submódulos de *Cadastro de Veículos* e *Cadastro de Condutores*, atividades que acabaram sendo alocadas para outros colegas de equipe. No lugar destes foram implementados *Cadastro de Locadores* e *Cadastro de Condutores*. Tais alterações foram originadas pelo chefe do departamento de desenvolvimento.

Outro ponto alterado foi o modo como as atividades seriam procedidas, de acordo com o cronograma seriam desempenhadas sequencialmente, entretanto, foi discutido em reunião e se decidiu por implementa-lás paralelamente, uma vez que essa prática não afetaria o resultado final do módulo e proporcionaria mais agilidade e produtividade ao desenvolvimento.

A tarefa de integração com os submódulos *Cadastro de Servidores*, *Cadastro de Órgãos* e *Catálogo de Materiais Bens e Serviços* não foi realizada, pois estes submódulos pertencem a um macromódulo que ainda não foi finalizado.

A metodologia de desenvolvimento estipulada no plano de estágio (Apêndice A) foi Scrum, porém não foi possível aplica-lá, pelo fato de que algumas práticas da mesma não puderam ser seguidas, como por exemplo, Daily Scrum, que são reuniões diárias que tem como objetivo disseminar o conhecimento sobre o que foi feito no dia

anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia. Ao invés do Scrum foi usada a metodologia eXtreme Programming (XP), que se mostrou mais adequada para a realidade de desenvolvimento enfrentada, devido a sua maior praticidade de aplicação.

No Quadro 01, estão listadas as tarefas executadas, especificando o responsável por cada uma delas, assim como a quantidade de horas necessárias para a execução.

Quadro 01 – Atividades realizadas.

Atividade	Descrição da atividade	Executor	Horas
1	Elicitação, especificação e validação de requisitos junto ao chefe do departamento de desenvolvimento.	Equipe	30h
2	*Estudo das tecnologias* JSF, Hibernate, Spring, Servlets.	Equipe	30h
3	*Configuração do* ambiente *Linux*, Eclipse, *TomCat* e do controle de versão.	Equipe	25h
4	Desenvolvimento de uma entidade para familiarização com a estrutura do código.	Equipe	35h
5	Desenvolvimento do submódulo Cadastro de Locadores.	*Individual*	55h
6	Desenvolvimento do submódulo Cadastro de Condutores.	*Individual*	25h
7	Integração com o submódulo Cadastro de Veículos.	*Individual*	15h
8	Integração com o submódulo Cadastro de Locadores	Individual	10h
9	*Suporte técnico*	*Equipe*	*70h*
		Total	**300h**

4.1 Detalhamento das atividades

A seguir, serão descritas com maior nível de detalhes as tarefas desempenhadas durante o período de estágio integrado.

4.1.1 Elicitação, especificação e validação de requisitos

Foi realizado um levantamento de requisitos para o macromódulo de

Controle e Manutenção de Veículo, mediante entrevistas com os servidores do MPPB, para tentar extrair o máximo de informações sobre os atributos de cada submódulo, uma vez que estes servidores seriam os usuários finais do sistema. Esse processo foi realizado de maneira simplista e levou em torno de uma semana.

Foram conduzidos testes funcionais dos submódulos criados para verificar eventuais falhas no sistema. Esses testes foram executados pelos servidores e consistiam no uso livremente do sistema afim de encontrar possíveis problemas. A inspeção consistiu em compartilhar o código com os outros membros da equipe para verificar a existência de erros e sugerir melhorias no código.

Nenhuma metodologia mais rigorosa de testes foi adotada, pois o próprio departamento não dispõe de uma, portanto foi decido que tal tarefa não teria prioridade relevante.

4.1.2 Estudo das tecnologias

Para realizar o desenvolvimento do módulo proposto, foi preciso estudar novas tecnologias que já faziam parte dos outros macromódulos (Catálogo de Materiais Bens e Serviços, Licitações, Controle de Estoque, Sistema de Pesquisa de Mercado, Sistema Patrimonial de Bens Móveis e Imóveis, Módulo de Controle de Obras) do Sistema Aristóteles. Nenhuma destas tecnologias era familiar até então, assim esta etapa do estágio foi de grande contribuição para o engrandecimento do saber, uma vez que foi possível compreender o funcionamento das tecnologias e que papel cada uma exerceria na implementação do módulo.

4.1.3 Configuração dos aplicativos utilizados

Para iniciar o desenvolvimento do sistema foi preciso adequar as máquinas disponíveis ao mesmo ambiente de programação que era usado como padrão pela equipe do Ministério Público localizado em

João Pessoa, Paraíba.

Essa adequação foi uma das dificuldades encontradas no estágio, pelo fato de que apenas três máquinas encontravam-se disponíveis para uso dos quatro estagiários e dentre essas somente uma atendia ao requisito do sistema operacional que era de 64 bits e necessitava consequentemente de um processador de 64 bits. Diante desta situação, fez se necessário que cada estagiário, utilizasse seu próprio laptop, mediante autorização do chefe do departamento, para dar seguimento ao módulo, caso contrário a evolução do sistema ficaria bastante prejudicado.

Além do sistema operacional, Ubuntu 10.02, foi preciso configurar o IDE Eclipse, no qual era feito toda codificação do módulo; o Apache Tomcat, que foi usado como servidor local; e o Controle de versão (SVN) para manter toda a codificação em sincronia com todos com todos os membros da equipe.

4.1.4 Familiarização com a estrutura do código

Para obter familiarização com a estrutura do código do Sistema Aristóteles, foi realizada a implementação de uma entidade do macromódulo *Catálogo*. Essa tarefa envolveu a programação de funcionalidades básicas como adicionar, editar, detalhar, excluir e pesquisar, o que forneceu uma excelente visão da estrutura e funcionamento do módulo.

4.1.5 Desenvolvimento dos submódulos

O MP é uma instituição permanente, essencial à função jurisdicional do Estado, incumbindo-lhe a defesa da ordem jurídica, do regime democrático e dos interesses sociais e individuais indisponíveis. Infelizmente, o Departamento de Transportes deste órgão não possuía controle informatizado das despesas com serviços, compras, uso, locação, manutenção de veículos, consumo de pneus e

combustíveis o que resultava na maioria das vezes no planejamento inexato das despesas com os serviços supracitados.

Tal departamento é tido como peça fundamental no processo de gestão e manutenção dos veículos possibilitando ao membro do MPPB o deslocamento até onde os direitos do cidadão estão sendo desrespeitados.

Na Figura 04, mostra-se a página de *login* do macromódulo de Controle e Manutenção de Veículos. Todos os estagiários envolvidos nesse projeto utilizavam um único *login* e senha para ter acesso ao sistema, que era executado em servidor local (Apache Tomcat) e também acessado a partir de um banco de dados local (PostgreSQL).

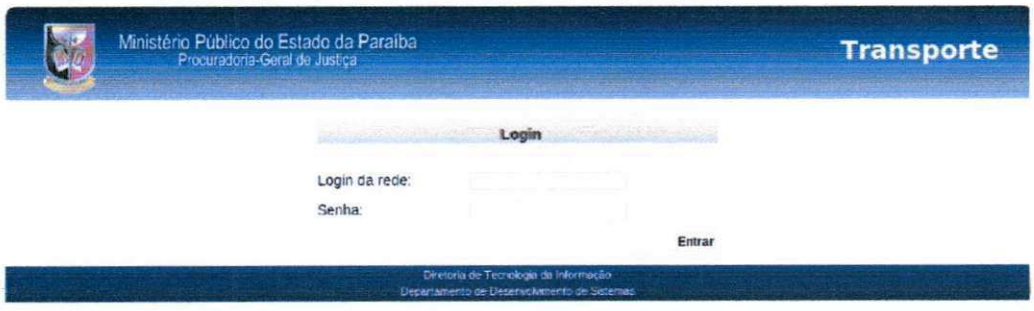

Figura 04 – Página de *login*.

Na Figura 05, tem-se o menu principal do sistema, constituído pelos submódulos *Cadastro de Combustíveis, Cadastro de Condutores, Cadastro de Fornecedores, Cadastro de Locadores, Cadastro de Veículos, Multas e Relatório.*

Figura 05 – Menu principal.

Os servidores constantemente precisam locar os veículos do

órgão tanto para se deslocarem dentro como fora da cidade, visto isso, se fez necessária a criação de um submódulo que mantivesse todos os registros dessas locações.

Cadastro de Locadores, que consiste no registro das atividades que fazem uso dos veículos do MPPB. Ele é composto da seguinte forma:

- Nome e matrícula do servidor que deseja fazer a locação;
- Local de onde a locação é feita e para onde o transporte seguirá;
- Datas de saída e chegada da locação;
- Veículo e condutor que serão alocados para a tarefa;
- O razão pela qual a locação se faz necessária.

A distribuição dos atributos supradescritos pode ser visualizada na Figura 06.

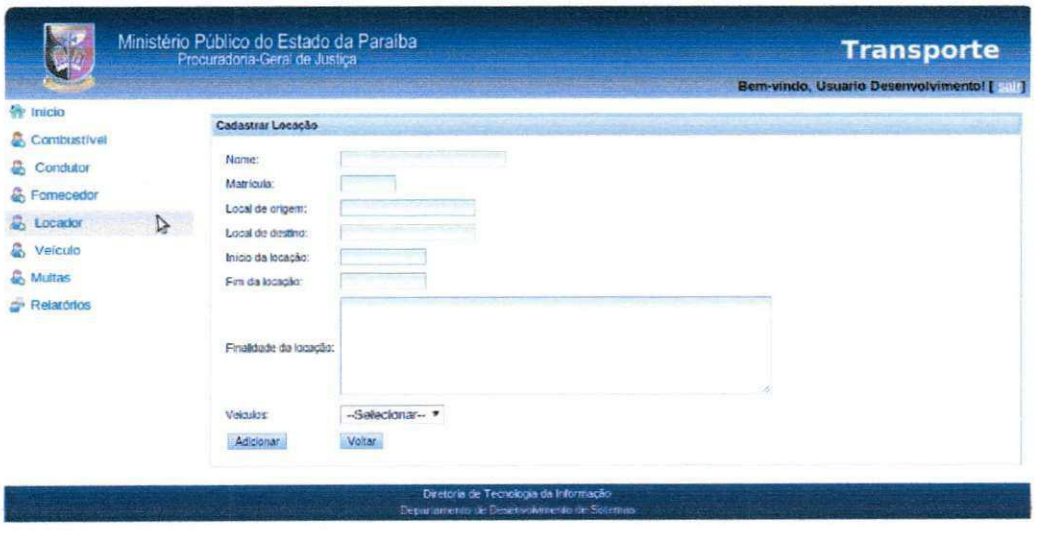

Figura 06 – Cadastro de Locador.

Na Figura 07 apresenta-se a página inicial do *Cadastro de Locadores*. A partir dela é possível ter acesso às seguintes operações:

- **Adicionar**: realiza o cadastro de uma nova locação, usando como parâmetro de entrada os atributos visualizados na Figura 06;
- **Detalhar**: mostra todos os atributos registrados de

determinada locação;

- **Editar**: permite realizar alterações nos dados de uma determinada locação;
- **Excluir**: deleta todos os dados de uma locação;
- **Pesquisar**: faz uma busca no banco de dados pelo nome do servidor digitado e retorna na tela todas as locações feitas por ele.

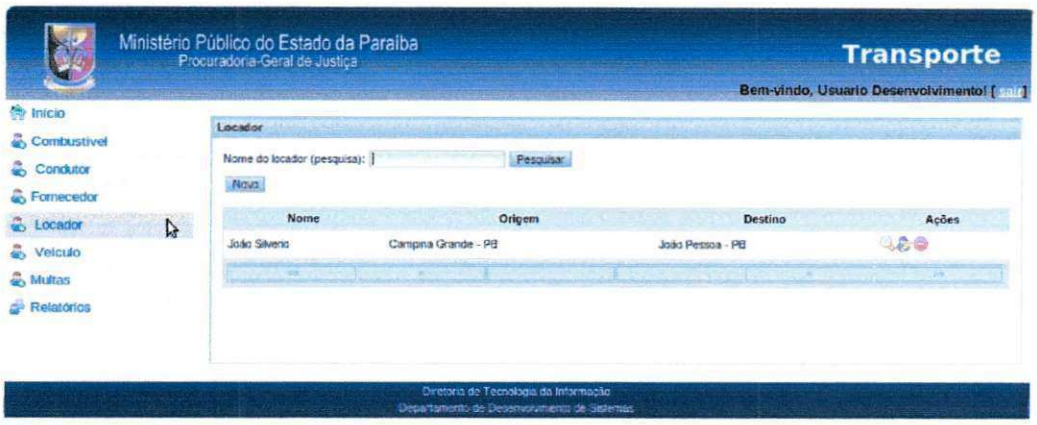

Figura 07 – Página inicial de Cadastro de Locadores.

Qualquer atividade que envolva os veículos do MPPB deve contar com a participação de um servidor devidamente habilitado para realizar a condução do transporte adequadamente.

O submódulo *Cadastro de Condutores* foi criado para registrar todos os servidores que são responsáveis por atuar como condutores do órgão. Ele é composto da seguinte forma:

- Nome e matrícula do condutor;
- Data de nascimento do condutor;
- Tipo de habilitação;
- Data de vencimento da habilitação;
- Número da habilitação.

A composição do mesmo pode ser observada na Figura 08:

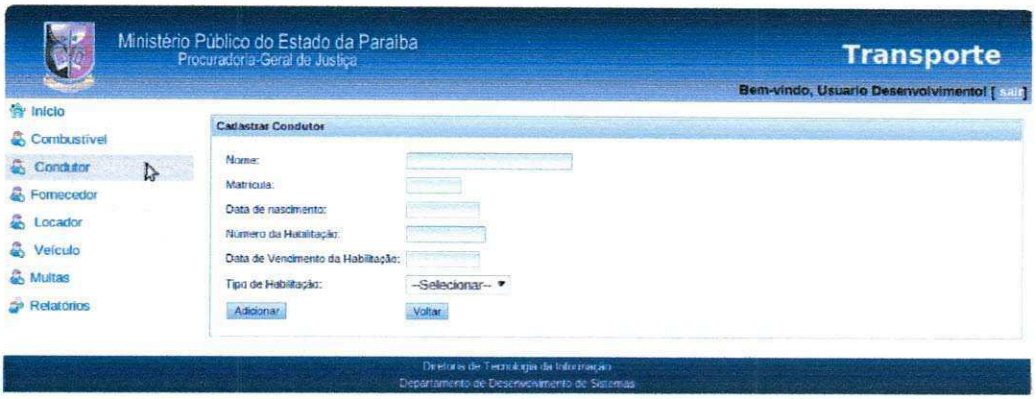

Figura 08 – Cadastro de Condutor.

A página de *Cadastro de Condutores* é mostrada na Figura 09. Suas funcionalidades são semelhantes àquelas do *Cadastro de Locadores*, detalhadas anteriormente.

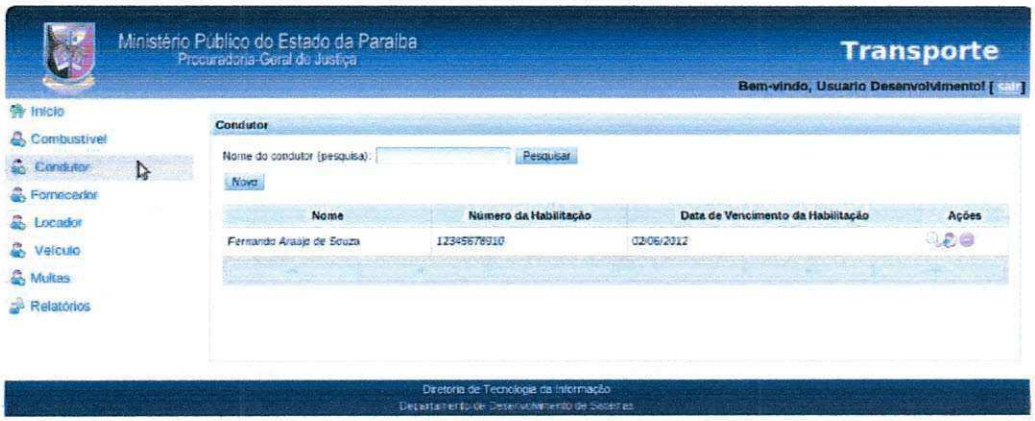

Figura 09 – Página inicial de Cadastro de Condutores.

4.1.6 Integração dos submódulos

Após a conclusão do submódulo Cadastro de Locadores, foi preciso realizar a integração do mesmo com o submódulo Cadastro de Veículos, uma vez que para realizar uma locação é preciso verificar a disponibilidade dos veículos cadastros no banco de dados.

Uma segunda integração se mostrou também necessária, desta vez entre Cadastro de Locadores e Cadastro de Condutores, uma vez que para realizar uma locação é preciso verificar a disponibilidade dos condutores cadastros no banco de dados.

4.1.7 Suporte técnico

As atividades relacionadas ao suporte técnico eram as mais recorrentes, ocorriam diariamente e isso exigia constante revezamento entre o papel de desenvolvedor e técnico.

Frequentemente, as máquinas do MPPB apresentavam algum tipo de problema de *hardware*, como por exemplo computadores que não conseguiam inicializar seus sistemas operacionais devido a problema na memória principal, ou de *software*, e.g., DLL do *Windows* que se corrompiam.

A manutenção da rede interna era uma das tarefas que exigiam mais agilidade na resolução, pois as impressoras em rede constantemente precisavam ser utilizadas para a impressão de documentos, principalmente no âmbito criminal, portanto, devido a problemas no sistema operacional de algumas máquinas ocasionalmente esses dispositivos deixavam de operar corretamente.

O suporte ao usuário no uso dos aplicativos adotados pelo MPPB era constante, pois o órgão conta com a presença de diversos servidores, que na maioria das vezes por possuírem idade mais avançada, não dispunham de grande habilidade para o manuseio dos programas, como por exemplo, o BrOffice.

A seguir, são listadas outras atividades que faziam parte do dia a dia do papel de técnico:

- Reciclagem de cartuchos;
- Detecção e remoção de vírus em geral;
- Limpeza interna dos computadores e periféricos;
- Backup de arquivos da instituição em CD/DVD;
- Configuração de periféricos.

SEÇÃO V

CONSIDERAÇÕES FINAIS

5 Considerações finais

O estágio é um processo de aprendizagem indispensável a qualquer profissional que deseja estar preparado para enfrentar os desafios de uma carreira, principalmente aqueles da área de Tecnologia da Informação. Participar deste processo foi uma chance de assimilar a teoria e a prática, aprender as peculiaridades e dicas da profissão, conhecer a realidade do dia a dia.

Nestes últimos meses, foi possível extrair uma grande diversidade de informação a partir das tarefas realizadas, principalmente durante o desenvolvimento do sistema abordado neste relatório, uma vez que as tecnologias usadas não faziam parte do domínio de conhecimento até o início do período.

Apesar de toda a gratidão pela chance de engrandecimento profissional mediante este estágio, não há como não mencionar o sentimento de subutilização, uma vez que o potencial como desenvolvedor poderia ter sido aproveitado de forma mais desafiadora.

Outro aspecto que deixou a desejar foi a grande quantidade de burocracia na execução das tarefas relacionada ao suporte técnico. A reposição de peças defeituosas se tornava impraticável, uma vez que para isso era preciso abrir licitações, e estas por sua vez só eram lançadas quando se atingia um número alto de reposições necessárias.

No somatório de tudo, com certeza o resultado pode ser considerado extremamente positivo. No âmbito pessoal, foi gratificante ter a chance de trabalhar com os colegas de curso, que se mostraram competentes e dedicados. No profissional, ter tido a oportunidade de aprender e aprimorar novas técnicas e tecnologias de programação e suporte ao usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

REFERÊNCIAS BIBLIOGRÁFICAS

CARVALHO, Marlon. **Spring Framework**. Disponível em: <http://imasters.com.br/artigo/4497/spring_framework_introducao> . Acessado em: 02 nov. 2010.

LARA, Sílvio Garbes. **Tutorial de programação Java**. Disponível em: <http://sites.google.com/site/silviogarbes/desenvolvimento-web/tutorial-de-programacao-java>. Acessado em: 10 nov. 2010.

OLIVEIRA, Alison Cardoso. **Tecnologia java para desktop utilizando Hibernate**. Disponível em: <http://web.unipar.br/~seinpar/artigos/alison-cardoso-de-oliveira.pdf>. Acessado em: 08 nov. 2010.

PITANGA, Talita. **JavaServer Faces: A mais nova tecnologia Java para desenvolvimento Web**. [S.L]. Disponível em: <http://www.guj.com.br/content/articles/jsf/jsf.pdf>. Acessado em: 12 nov. 2010.

SILVA, Jackson Luiz. **Banco de dados comunitário**. Disponivel em: <http://projetos.inf.ufsc.br/arquivos_projetos/projeto_635/TCC-Jackson-Final-BDC.pdf>. Acessado em: 07 nov. 2010.

VALENTIM, Ricardo A. M. **Eclipse: uma IDE Para Java**. Disponível em: http://www.cefetrn.br/~valentim/disciplinas/poo/material/Eclipse_Tutorial.pdf>. Acessado em: 05 nov. 2010.

APÊNDICES

APÊNDICE A - PLANO DE ESTÁGIO

UFCG - UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CEEI - CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DSC - DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

Plano de Estágio Integrado

Ministério Público

Desenvolvimento do macromódulo de Controle e Manutenção de Veículos do Ministério Público da Paraíba

Antonio Alexandre Moura Costa

Mat.: 20921282

Prof. Dr. José Eustáquio Rangel de Queiroz

Orientador Acadêmico

Campina Grande,

Setembro 2010

Informações Pessoais

Nome: Antonio Alexandre Moura Costa

Curso: Ciência da Computação

Matricula: 20921282

CPF: 048.753.964.81

RG: 2923843 SSP/PB

Endereço: Rua Malaquias Souza do Ó, nº 34, Mirante

Campina Grande – PB

CEP: 58407563

Email: alexandrecostapb@gmail.com

Ambiente de Estágio

O estágio será realizado no 2ª Centro de Apoio Operacional às Curadorias, situada à Rua Promotora Terezinha Lopes de Moura S/N, Liberdade, 58.410-064, Campina Grande, Paraíba.

O trabalho será realizado por uma equipe de 5 (cinco) pessoas, sendo quatro desenvolvedores e um gerente, na qual a atuação se dará como desenvolvedor.

Supervisão

Supervisor Acadêmico

Nome: Prof. José Eustáquio R. de Queiroz, D.Sc.

Endereço: Departamento de Sistemas e Computação

Universidade Federal de Campina Grande

Avenida Aprígio Veloso, 882 – CEP: 58.109-970

Bodocongó, Campina Grande, PB – Brasil.

Email: rangeldequeiroz@gmail.com

Supervisor Técnico

Nome: Uirá Vasconcelos Alencar de Assis

(Chefe de Suporte Técnico do Ministério Público da Paraíba)

Endereço: Rua Promotora Terezinha Lopes de Moura, S/N

CEP: 58.410-064

Liberdade, Campina Grande, PB – Brasil.

Email: uira@mp.pb.gov.br

Resumo do Problema do Estágio

O Ministério Público da Paraíba é um órgão da esfera jurídica cuja atuação acontece junto ao cidadão paraibano atuando na defesa dos seus direitos. O Departamento de Transportes do MPPB é de vital importância nesse processo de gestão e manutenção dos veículos possibilitando ao membro do MPPB o deslocamento até onde os direitos do cidadão estão sendo desrespeitados.

Atualmente o Departamento de Transportes não possui controle informatizado das despesas com serviços, locação, manutenção de veículos, consumo de pneus e combustíveis o que resulta na maioria das vezes no planejamento inexato das despesas com os serviços citados acima.

É neste contexto que é inserido o Aristóteles, conjunto de ações e sistemas, automatizados e integrados, que tem três grandes objetivos:

- O controle da despesa administrativa no MPPB e a eliminação de desperdícios através de um melhor planejamento dessas despesas;
- Valorização e qualificação da atividade do servidor do MPPB;
- Eficiência e eficácia na prestação de serviços à sociedade.

O Sistema Aristóteles é divido em 7(sete) macromódulos:

1. Catálogo de Materiais Bens e Serviços (Em processo de implantação);
2. Licitações (Não Iniciado);
3. Controle de Estoque (Em desenvolvimento);
4. Sistema de Pesquisa de Mercado (Em desenvolvimento);
5. Sistema Patrimonial de Bens Móveis e Imóveis
6. Módulo de Controle de Obras (Não iniciado);
7. Módulo Controle e Manutenção de Veículos (Não iniciado).

Objetivo do estágio

O objetivo do estágio é desenvolver e implantar parte do macromódulo de Controle e Manutenção de Veículos do Departamento de Transportes do MPPB.

Os submódulos a serem desenvolvidos durante o período de estágio, estão especificados nas seções de Atividades planejadas e cronograma.

Já as tecnologias a serem utilizadas serão definidas durante o estágio.

Metodologia

O Departamento de Desenvolvimento de Sistemas do MPPB, subordinado à Diretoria de Tecnologia da Informação, está utilizando, para o desenvolvimento de seus sistemas, uma metodologia ágil com práticas de SCRUM [1] e Kanban para acompanhamento e transparência das tarefas (*user stories* ou requisitos do sistema) atribuídas ao desenvolvedor, como também práticas de RUP (*Rational Unified Process*) [2] para coleta e documentação dos sistemas.

Basicamente, o SCRUM é um processo de desenvolvimento iterativo e incremental para gerenciamento de projetos e desenvolvimento ágil de *software*. O Kanban sinaliza um controle de fluxo das atividades via registros, permitindo sinalização imediata da demanda que sustenta as prerrogativas da cadeia de elaboração, inspeção e execução de um processo. Como mencionado, técnicas de RUP estão sob utilização, tendo como pilar a redução do risco (existente em qualquer projeto de *software*), tornando mais eficiente o desenvolvimento com o controle de mudanças e a verificação contínua de qualidade.

Estes processos darão apoio às etapas do projeto descritas no Quadro 01

Quadro 01 – Síntese da metodologia a ser adotada.

Etapa	*Descrição*	*Executor*
Levantamento de requisitos	Descrição dos requisitos aplicáveis ao sistema junto ao cliente	Gerente
Cronograma detalhado de trabalho	Definição detalhada da ordem e período de cada etapa do desenvolvimento	Gerente
Planejamento de releases	Planejamento do projeto de desenvolvimento como um todo em alto nível	Gerente
Modelagem da aplicação	Definição dos relacionamentos entre os módulos	Gerente/Estagiários
Estimativas de custo	Planejamento do tempo utilizado em cada atividade	Estagiários/Gerente
Implementação	Geração de código	Estagiários
Testes de unidade e ajustes na ferramenta	Geração de testes e correções	Estagiários

Atividades Planejadas

Devem ser desenvolvidas as seguintes atividades no estágio:

I. Desenvolver e documentar o Módulo de Controle e Manutenção de Veículos do Departamento de Transportes do MPPB.

1. Objetivos do sistema:

a) Racionalizar e controlar os recursos financeiros nas despesas com serviços, compras, uso, locação e manutenção de veículos, consumo de pneus e combustíveis;

b) Disciplinar a implantação, manutenção e atualização do cadastro de veículos próprios do MPPB e a ele locados ou cedidos;

c) Treinar o servidor na nova filosofia do controle e acompanhamento da atividade administrativa através da avaliação de desempenho e de indicadores de qualidade e produtividade definidos no Planejamento Estratégico do MPPB.

2. Módulos do sistema de Controle e Manutenção de Veículos:

a) Manutenção de Fornecedores;

b) Cadastro de Servidores;

c) Manutenção de Locadores;

d) Manutenção de Órgãos;

e) Cadastro dos Veículos;

f) Produtos;

g) Combustível;

h) Lubrificantes;

i) Pneus;

j) Baterias;

k) Manutenção;

l) Alinhamento/Balanceamento.

Estarei envolvido ativamente no desenvolvimento dos módulos de *Cadastro dos Veículos* e *Cadastro de Combustíveis* e exercerei no papel de testador dos módulos, *Cadastro de Fornecedores* e *Cadastro de Locadores*.

Cronograma de atividades

O cronograma de atividades proposto é o seguinte:

Tarefa	*Set*	*Out*	*Nov*	*Dez*
Estudo da arquitetura do sistema e tecnologias utilizadas.	X			
Cadastro de Fornecedores.	X			
Integração com o Cadastro de Servidores.		X		
Cadastro de Locadores.		X		
Integração com o Cadastro de Órgãos.			X	
Cadastro dos Veículos.			X	
Integração com o Catálogo de Materiais Bens e serviços para o sub-módulo de manutenção de produtos.			X	
Cadastro de Combustíveis.				X
Testes e integração com os módulos existentes.	X	X	X	X
Escrita de relatório técnico	X	X	X	X
Defesa do Estágio				X

Resultados esperados

Após a implantação, treinamento e correta alimentação do sistema por parte dos servidores do Departamento de Transportes esperamos fornecer condições para que o chefe do departamento possa realizar um planejamento mais exato das despesas de transporte e consequente redução dos desperdícios.

Bibliografia

[1] SCHWABER, K., Agile Project Management with Scrum ed. Microsoft Press, 2004. ISBN 978-0-735-61993-7.

[2] KRUCHTEN, P., Introdução ao RUP : Rational Unified Process ed. Ciência Moderna, 2003. ISBN 8-57393-275-9

Aprovação

Uirá Vasconcelos Alencar de Assis

Supervisor Técnico

José Eustáquio R. de Queiroz

Orientador Acadêmico

Joseana Macêdo Fechine

Coordenadora da disciplina Estágio Integrado

ANEXO A - CÓDIGO DA CLASSE LOCADORMBEAN.JAVA

```
package mppb.transporte.view.backingbean;

import java.util.ArrayList;

import java.util.List;

import javax.faces.event.ActionEvent;

import mppb.transporte.model.entities.Locador;

import mppb.transporte.model.entities.Veiculo;

import mppb.transporte.model.facade.TransporteFacade;

import
org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Controller;

import br.gov.pb.mp.framework.persistence.util.SearchMode;

import
br.gov.pb.mp.sistemabase.view.backingbean.AbstractCrudBean;


@Controller

@Scope("session")

public class LocadorMBean  extends
AbstractCrudBean<Locador> {

    private static final long serialVersionUID = -
7517051455375425914L;

    private List<Locador> locadores = new
ArrayList<Locador>();

    private Locador locador = new Locador();

    private Locador filtro = new Locador();

    @Autowired

    private TransporteFacade transporteFacade;

    @Override

    public void doSalvar() throws Exception {
```

```
        transporteFacade.salvarLocador(locador);
    }
    @Override
    public List<Locador> doPesquisar() throws Exception {

        filtro._getPersistence().setPaged(true);
        filtro._getPersistence().addOrderBy("nome");
        filtro._getPersistence().addLikeProperty("nome",
              SearchMode.ANYWHERE_IGNORE_CASE);
        return transporteFacade.pesquisarLocador(filtro);
    }
    @Override
    protected void initFilter(Locador entidade, Long id) {
        if(entidade == null){
            filtro = new Locador();
        } else {
            filtro = entidade;
        }

        if(id != null){
            filtro.setId(id);
        }
    }
    @Override
    protected void initEntity(Locador entidade, Long id) {
        if(entidade == null){
            locador = new Locador();
        } else {
            locador = entidade;
        }
```

```
            if(id != null){
                locador.setId(id);
            }
        }
        public String manter(){
            String page = super.manter();
            initFilter(null, null);
            return page;
        }
        @Override
        public void doExcluir() throws Exception {
            transporteFacade.excluirLocador(locador);
        }
        @Override
        public String getEntityName() {
            return "locador";
        }
        @Override
        protected void initEntidades(List<Locador>
listaEntidades) {
            this.locadores = listaEntidades;
        }
        @Override
        protected String getManterNavigationRule() {
            return "page.manter_locador";
        }
        public List<Locador> getLocadores() {
            return locadores;
        }
        public void setLocadores(List<Locador> locadores) {
```

```
        this.locadores = locadores;
    }
    public Locador getLocador() {
        return locador;
    }
    public void setLocador(Locador locador) {
        this.locador = locador;
    }
    public Locador getFiltro() {
        return filtro;
    }
    public void setFiltro(Locador filtro) {
        this.filtro = filtro;
    }
    public void actionListener(ActionEvent event){
        super.actionEvent(event, locadores);
    }
    public List<Veiculo> getSelectVeiculos(){
        return null;
    }
}
```

ANEXO B - CÓDIGO DA CLASSE LOCADOR.JAVA

```
package mppb.transporte.model.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.SequenceGenerator;

import javax.persistence.Table;

import mppb.catalogo.model.entities.Natureza;

import mppb.publico.model.util.Constants;

import org.hibernate.annotations.Where;

import
br.gov.pb.mp.framework.persistence.entities.AbstractEntity;

import
br.gov.pb.mp.framework.persistence.entities.IEntidadeExclui
velLogicamente;


@Table(name = "locador", schema =
Constants.TRANSPORTE_SCHEMA)

@Entity

@SequenceGenerator(name = "sequence", sequenceName =
Constants.TRANSPORTE_SCHEMA

          + ".locador_id_seq")

@Where(clause = "excluido='f'")

public class Locador extends AbstractEntity implements
```

```
        IEntidadeExcluivelLogicamente {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO,
generator = "sequence")
    @Column(name = "id")
    private Long id;
    @Column(name = "nome")
    private String nome;
    @Column(name = "origem")
    private String origem;
    @Column(name = "destino")
    private String destino;
    @Column(name = "finalidade")
    private String finalidade;
    @ManyToOne( fetch = FetchType.LAZY)
   @JoinColumn(name="id_veiculo")
    private Veiculo veiculo;
    @Column(name = "matricula")
    private Long matricula;
    @Column(name = "inicio")
    private String inicio;
    @Column(name = "fim")
    private String fim;
    @Column(name = "excluido")
    private Boolean excluido = false;
    @Override
    public Long getId() {
        return id;
    }
```

```
@Override
public void setId(Long id) {
    this.id = id;
}
@Override
public Boolean getExcluido() {
    return excluido;
}
@Override
public void setExcluido(Boolean excluido) {
    this.excluido = excluido;
}
public String getNome() {
    return nome;
}
public void setNome(String nome) {
    this.nome = nome;
}
public String getOrigem() {
    return origem;
}
public void setOrigem(String origem) {
    this.origem = origem;
}
public String getDestino() {
    return destino;
}
public void setDestino(String destino) {
    this.destino = destino;
}
```

```
public String getFinalidade() {
     return finalidade;
}
public void setFinalidade(String finalidade) {
     this.finalidade = finalidade;
}
public Veiculo getVeiculo() {
     return veiculo;
}
public void setVeiculo(Veiculo veiculo) {
     this.veiculo = veiculo;
}
public Long getMatricula() {
     return matricula;
}
public void setMatricula(Long matricula) {
     this.matricula = matricula;
}
public String getInicio() {
     return inicio;
}
public void setInicio(String inicio) {
     this.inicio = inicio;
}
public String getFim() {
     return fim;
}
public void setFim(String fim) {
     this.fim = fim;
}
```

ANEXO C - CÓDIGO DA CLASSE LOCADORSERVICE.JAVA

```
package mppb.transporte.model.service;

import java.util.List;

import mppb.transporte.model.entities.Locador;

import
org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Service;

import
br.gov.pb.mp.sistemabase.persistence.service.PersistenceSer
vice;


@Service

@Scope("singleton")

public class LocadorService {


     @Autowired

     private PersistenceService<Locador>
locadorPersistence;


     public List<Locador> pesquisarLocador(Locador
locador){

          return locadorPersistence.pesquisar(locador);

     }


     public Locador pesquisarUmLocador(Locador locador){

          return locadorPersistence.pesquisarUm(locador);

     }


     public Long salvarLocador(Locador locador){

          return locadorPersistence.salvar(locador);
```

```
    }

    public void excluirLocador(Locador locador){
        locadorPersistence.excluir(locador);
    }
}
```