UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JARIEL GUILHERME LUVIZON

SEGURANÇA E DESEMPENHO EM APLICAÇÕES WEB UTILIZANDO JAAS, GLASSFISH E POSTGRESQL

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

JARIEL GUILHERME LUVIZON

SEGURANÇA E DESEMPENHO EM APLICAÇÕES WEB UTILIZANDO JAAS, GLASSFISH E POSTGRESQL

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. M.Sc. Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Segurança e desempenho em aplicações WEB utilizando JAAS, GlassFish e PostgreSQL

Por

Jariel Guilherme Luvizon

Este Trabalho de Diplomação (TD) foi apresentado às 10:30 h do dia 14 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. M.Sc. Fernando Schütz UTFPR – *Campus* Medianeira (Orientador)	Prof. Cláudio Leones Bazzi UTFPR – *Campus* Medianeira (Convidado)
Prof. Márcio Angêlo Matté UTFPR – *Campus* Medianeira (Convidado)	Prof. Juliano Lamb UTFPR – *Campus* Medianeira (Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Agradeço primeiramente a toda minha família que sempre me apoiou e incentivou durante os momentos mais difíceis e deram conselhos que me fizeram superar obstáculos e seguir em frente com o trabalho.

Agradeço também aos meus amigos, em especial ao Felipe e ao Leandro, com os quais eu passei vários dias trabalhando no desenvolvimento do sistema.

Esse trabalho com certeza não seria possível sem a ajuda de todos os professores que participaram da minha caminhada pela UTFPR, com os quais eu aprendi muito. Guardo uma gratidão em especial a meu orientador Fernando Schütz que me orientou tanto no estágio quanto no TCC e sempre esteve disposto a ajudar, apesar das dificuldades.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.” (EINSTEIN, Albert)

RESUMO

LUVIZON, G. Jariel. Segurança e desempenho em aplicações *WEB* utilizando JAAS, GlassFish e PostgreSQL. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

A segurança e o desempenho nas aplicações *WEB* é algo fundamental. Este trabalho tem como foco demonstrar de maneira teórica e com exemplos práticos como deixar uma aplicação *WEB* segura, através do controle de autenticação e autorização utilizando JAAS (*Java Authentication and Authorization Service*) e explorando os recursos de segurança e *backup* fornecidos pelo SGBD (Sistema Gerenciador de Banco de Dados) PostgreSQL. Também será aprimorado a performance do servidor de aplicação GlassFish para melhorar o desempenho da aplicação *WEB.*

Palavras-chave: JAAS. GlassFish. PostgreSQL. Segurança. Desempenho.

ABSTRACT

LUVIZON, G. Jariel. Segurança e desempenho em aplicações WEB utilizando JAAS, GlassFish e PostgreSQL. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

The safety and performance in WEB applications is fundamental. This work focuses on theoretical and demonstrate the way with practical examples how to make WEB application security by controlling authentication and authorization using JAAS (Java Authentication and Authorization Service) and exploiting the resources of security and backup provided by the DBMS (System Manager Database) PostgreSQL. Will also be improved performance of the application server GlassFish to improve performance of WEB application.

Keywords: JAAS. GlassFish. PostgreSQL. Safety. Performance.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ACID	-	Atomicidade, Consistência, Isolamento e Durabilidade
API	-	*Application Programming Interface*
AVI	-	*Audio Video Interleave*
BMP	-	*BitMap*
CAPTCHA	-	*Completely Automated Public Turing Test to Tell Computers and Humans Apart*
CSS	-	*Cascading Style Sheets*
CDDL	-	*Common Development and Distribution License*
EJB	-	*Enterprise JavaBean*
GB	-	*GigaByte*
GIF	-	*Graphics Interchange Format*
HTML	-	*HyperText Markup Language*
HTTP	-	*Hypertext Transfer Protocol*
ISO	-	*International Organization for Standardization*
JAAS	-	*Java Authentication and Authorization Service*
JDBC	-	*Java Database Connectivity*
JEE	-	*Java Enterprise Edition*
JPEG/JPG	-	*Joint Photographic Experts Group*
JS	-	*JavaScript*
JSF	-	*Java Server Faces*
MVCC	-	*Multiversion Concurrency Control*
ODBC	-	*Open Data Base Connectivity*
OS	-	*Operating System*
PAM	-	*Pluggable Authentication Module*
PITR	-	*Point-in-time Recovery*
PNG	-	*Portable Network Graphics*
SDK	-	*Software Development Kit*
SGBD	-	Sistema Gerenciador de Banco de Dados
SQL	-	*Structured Query Language*
TB	-	*TeraByte*
URL	-	*Uniform Resource Locator*

XML	-	*Extensible Markup Language*
XHTML	-	*eXtensible Hypertext Markup Language*
WAL	-	*Write Ahead Logs*

SUMÁRIO

1 INTRODUÇÃO

O acesso a Internet tornou-se algo praticamente indispensável no mundo atual. Transações bancárias, *e-commerce,* estudo a distancia e outras infinidades de recursos estão disponíveis na Rede. E como não poderia ser diferente de outros aspectos da vida, a segurança é fundamental e deve ser uma das principais preocupações na hora de desenvolver uma aplicação *WEB.*

Uma aplicação típica, geralmente, está distribuída em vários servidores, rodando diversos aplicativos e para funcionar na velocidade adequada, a aplicação precisa de que as interfaces entre os diversos sistemas sejam construídas com a premissa que os dados passados através da mesma são confiáveis e não hostis. O "calcanhar de Aquiles" destas aplicações é a necessidade de haver "confiança" entre os diversos subsistemas e é disso que os *hackers* e outros "cibercriminosos" se aproveitam (GARTNER, 2004).

Entretanto, a segurança de uma aplicação *WEB* não se baseia somente em proteger-se de ataques externos e de controlar o acesso de usuários autenticados. Além de tudo isso a aplicação deve ser capaz de manter a integridade dos dados dos usuários e garantir que eles permaneçam intactos independente do que ocorra.

Aliado a segurança, outro fator importante em uma aplicação *WEB* é o desempenho. De nada adianta uma aplicação ser segura em vários níveis se a mesma não tiver um desempenho que agrade o usuário.

1.1 OBJETIVO GERAL

Este trabalho objetiva desenvolver uma aplicação com a finalidade de testar o nível de segurança e o desempenho de aplicações *WEB* utilizando o serviço de autenticação e segurança em Java (JAAS), o servidor de aplicação GlassFish e o banco de dados PostgreSQL.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

- Realizar uma pesquisa bibliográfica sobre as ferramentas JAAS, GlassFish e PostgreSQL;
- Desenvolver uma aplicação *WEB* Java EE (*Enterprise Edition)*.
- Implementar o mecanismo de segurança JAAS na aplicação;
- Apresentar um esquema que vise a otimização de desempenho e segurança dos dados explorando os recursos oferecidos pelo servidor de aplicação GlassFish e do SGBD PostgreSQL.
- Testar na prática o desempenho e a segurança da aplicação após a configuração dos mecanismos de otimização de performance e segurança fornecidos pelo GlassFish.

1.3 JUSTIFICATIVA

Existem várias maneiras de fazer o controle de autorização e autenticação de usuários em uma aplicação *WEB*. O JAAS (Java *Authentication and Authorization Service*) é um conjunto de APIs que permite que as aplicações Java tenham um controle de autenticação e de acesso. O JAAS implementa uma versão Java do *framework* padrão *Pluggable Authentication Module* (PAM), e suporta autorização baseada em usuário. Isso permite que aplicações fiquem independentes desse controle de segurança. Serve para controlar permissões de vários tipos de recursos: arquivos, diretórios, conteúdos, URLs (VIANA, 2006).

Como esse padrão de segurança é executado em nível de servidor de aplicação é importante utilizar um que forneça diversos recursos e que possua um bom desempenho. O GlassFish é um servidor de aplicação Java EE desenvolvido pela Sun Microsystems. Ele fornece toda uma infraestrutura de serviços como, por exemplo, a criação de *pool* de conexões ao banco de dados e ao *deploy* de várias aplicações no mesmo servidor (JEFFERSON, 2008).

A escolha do SGBD é um outro ponto importante pois é ele que irá guardar todos os dados dos usuários. O PostgreSQL é um poderoso sistema gerenciador de banco de dados objeto-relacional de código aberto . Ele possui vários recursos que não apenas garantem a integridade dos dados, como também auxiliam no desempenho da aplicação.

Unindo essas três ferramentas (JAAS, GlassFish e PostgreSQL) é possível desenvolver uma aplicação segura, estável e com um ótimo desempenho, podendo assim aumentar o nível de aceitação da aplicação perante ao usuário.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho possui cinco capítulos, sendo que o primeiro trata sobre a contextualização do tema abordado e também define os objetivos e a justificativa do projeto.

O segundo busca fornecer um conceito teórico sobre as tecnologias e ferramentas que foram utilizadas no desenvolvimento do mesmo, como o JAAS (Java *Authentication and Authorization Service*), o servidor de aplicação GlassFish e o Sistema Gerenciador de Banco de Dados PostgreSQL.

O capítulo seguinte contempla o estudo experimental, onde foi mostrado na prática a solução para os requisitos de segurança e desempenho da aplicação desenvolvida.

As conclusões finais sobre o projeto e as sugestões para trabalhos futuros são abordadas no capítulo quatro.

Por fim, no último capítulo, estão as referências bibliográficas que fizeram parte do embasamento teórico do projeto.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem como objetivo fornecer um referencial teórico sobre as tecnologias utilizadas para o desenvolvimento do estudo experimental, que são o banco de dados PostgreSQL, o servidor de aplicações GlassFish e o mecanismo de segurança JAAS. Também serão abordados conceitos sobre os temas segurança e desempenho em aplicações *WEB*.

2.1 SEGURANÇA EM APLICAÇÕES WEB

A utilização de aplicativos *WEB* é algo que, a cada dia que passa, se torna cada vez mais indispensável. A maioria das pessoas prefere acessar uma aplicação *on-line* que permite realizar as mesmas ou até mais tarefas que um aplicativo *desktop* fornece e isso tudo sem ter o transtorno de precisar instalar o programa no próprio computador (LINGHAM, 2007).

Tanto as aplicações *desktop* como as *WEB* precisam cuidar de algo que é essencial, a segurança. Uma das principais vantagens desta ultima que é poder ser acessada de qualquer lugar que tenha acesso à Internet, se torna um dos seus pontos fracos.

Aplicações *WEB* são formadas por inúmeros recursos, como páginas dinâmicas, estáticas, imagens, *downloads*, *uploads*, processos, relatórios etc. Grande parte desses recursos não podem estar disponível para qualquer pessoa que tentar acessá-lo e para isso as aplicações *WEB* devem controlar o acesso dos usuários a estes recursos (FRANZINI, 2009). Para que essas aplicações estejam seguras existem alguns princípios que devem ser considerados:

- Autenticação: O processo de provar a sua identidade para uma aplicação;
- O controle de acesso de recursos: Os meios pelos quais as interações com os recursos são limitados para os usuários, papéis (*roles*), ou programas com o objetivo de impor a integridade, confidencialidade, ou restrições de disponibilidade;

- A integridade dos dados: Os meios de provar que um terceiro não alterou informações enquanto a mesma estava em trânsito;
- Confidencialidade e privacidade dos dados: Os meios utilizados para assegurar que a informação feita é disponível somente para usuários que estão autorizados a acessá-lo (BROWN et al, 2005).

2.1.1 Autenticação

Desde o surgimento da Internet, ela se mostrou como um mecanismo de anonimato, onde as pessoas poderiam se passar por outras ou até mesmo não se passar por ninguém. Autenticação é o processo onde uma pessoa prova quem ela realmente é com o intuito de obter alguma informação ou executar alguma ação (FRANZINI, 2009).

A autenticação geralmente é feita em uma tela de *login* onde a pessoa informa suas credencias (geralmente usuário e senha) e caso forem corretas, ela tem sua autenticação confirmada pela aplicação.

2.1.2 Autorização

Autorização consiste na etapa de dar permissão à pessoa autenticada para acessar algum recurso ou realizar alguma ação. Em sistemas onde existem vários tipos de usuários um administrador de sistemas definirá quais usuários terão acesso e quais privilégios de execução esses terão. (FRANZINI, 2009).

Segundo Franzini (2009), O mecanismo de autorização é muitas vezes visto como composto de dois momentos:

- Ato de atribuir permissões ao usuário do sistema no momento de seu cadastro, ou posterior.
- Ato de checar as permissões que foram atribuídas ao usuário no momento de seu acesso.

2.2 JAAS

O Serviço de Autenticação e Autorização JavaTM (JAAS) foi introduzido como um pacote opcional (extensão) para o JavaTM 2 SDK, *Standard Edition* (J2SDK), v1.3. Ele também está integrado no J2SDK desde a versão 1.4. (JAAS REFERENCE GUIDE, 2001).

O JAAS é responsável por definir um padrão JEE de regras de como a aplicação *WEB* irá definir e gerenciar o controle de acesso. Há diferentes níveis em opções de autenticação que são baseados na definição de *roles* (FRANZINI, 2009).

A base de dados onde ficam armazenados os usuários credenciados pode estar em arquivos xml, bancos de dados relacionais, etc. (FRANZINI, 2009).

Utilizando o JAAS é possível simplificar o desenvolvimento de segurança de uma aplicação Java *WEB*, colocando uma camada de abstração entre o aplicativo e os diferentes mecanismos de autorização e autenticação. Essa independência de plataformas e algoritmos permite a utilização de mecanismos de segurança diferentes, sem modificar seu código em nível de aplicação (MUSSER et al, 2009).

O modulo de autenticação do JAAS está em nível de servidor de aplicação, por isso ele será executado por esse último, antes mesmo de acessar a aplicação (VIANA, 2006). A Figura 1 mostra o funcionamento do JAAS entre as camadas de uma aplicação *WEB.*

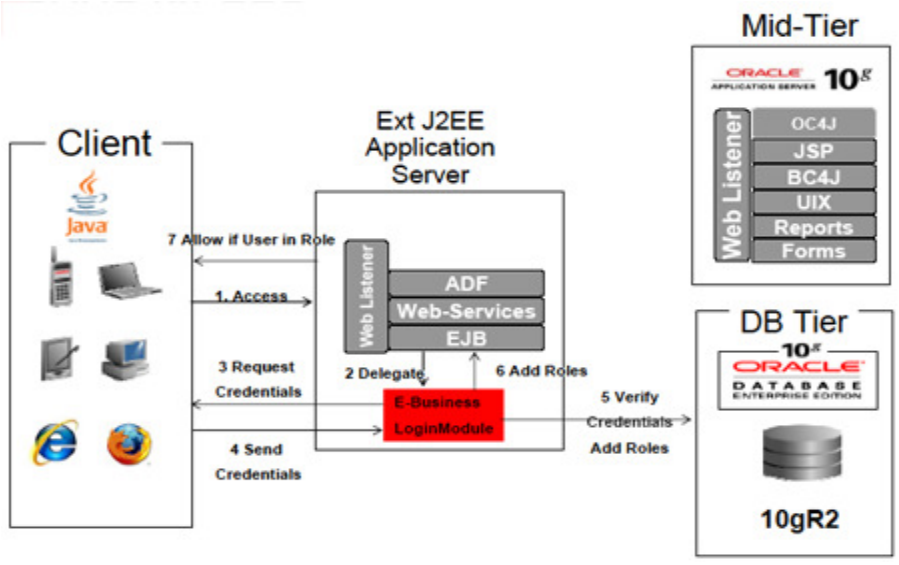

Figura 1 – Funcionamento do JAAS
Fonte: Oracle (2008)

2.3 SEGURANÇA EM UM SGBD

A segurança de um banco de dados é algo de extrema importância pois eles guardam um patrimônio muito valioso que é a informação. Quando há um problema de segurança em um SGBD são os dados que estão ameaçados, dados esses que podem ser senhas bancárias, números de cartões de credito, entre outras informações confidenciais.

Segurança em SGBD não se refere apenas ao banco de dados, diz respeito a toda a cadeia tecnologia associada à aplicação, onde um elo fraco põe em risco o conjunto todo. Para que isso não aconteça não se deve pensar no SGBD isoladamente. É necessário pensar no equipamento onde o SGBD se encontra, no OS onde ele está instalado, na própria instalação do SGBD, nas informações contidas no banco de dados, nas aplicações que vão acessar o banco de dados e também no usuário que precisa destas informações (TELLES, 2008).

Existem alguns aspectos que caminham lado a lado com o conceito de segurança em um SGBD: integridade, *backup* e controle de acesso.

- Integridade: engloba as propriedades ACID (Acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade). É necessário garantir que os dados contidos no banco de dados se mantenham intactos independente do que ocorra. Mesmo que aconteça algum problema de indisponibilidade durante um determinado tempo é fundamental que quando o problema seja resolvido os dados voltem exatamente como estavam. Para garantir a integridade também é fundamental usar as restrições de integridade (*primary key, foreign key, index*, etc), pois caso ocorra um erro do usuário ou da aplicação essas restrições vão impedir que os dados sejam corrompidos (TELLES, 2008).
- *Backup*: existe como forma de prevenir para quando os dados forem perdidos acidentalmente, seja por falha física, ou por falha humana, exista uma cópia desses dados. O *backup* garante a integridade dos dados, de configurações, de arquivos de usuários, etc. Os *backups* devem fazer parte da rotina de operação dos sistemas e seguir uma política determinada. É interessante faze-los de maneira automatizada para reduzir o seu impacto sobre o trabalho dos administradores e operadores de sistemas (RIBEIRO, 2009) .

- Controle de Acesso: Segundo Telles (2008), é a capacidade de que as informações disponíveis no banco de dados estejam disponíveis para as pessoas corretas, que terão permissão de acesso apenas às operações permitidas para o seu perfil. Normalmente o controle de acesso é feito a partir de objetos do banco de dados, porém é possível restringir o acesso, por exemplo, a apenas algumas linhas de uma tabela. Esse controle pode também limitar os recursos utilizados no banco de dados, como conexões ativas, memória, processador, etc (TELLES, 2008).

2.4 POSTGRESQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional de código aberto (POSTGRESQL, 2011). Ele foi baseado no POSTGRES Versão 4.2 desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley, o qual foi pioneiro em vários conceitos que somente se tornaram disponíveis muito mais tarde em alguns sistemas de banco de dados comerciais. (DOCUMENTAÇÃO DO POSTGRESQL 8.0, 2005).

O PostgreSQL Tem mais de 15 anos de desenvolvimento ativo e uma arquitetura que comprovadamente ganhou forte reputação de confiabilidade, integridade de dados e conformidade a padrões. Roda em todos os grandes sistemas operacionais, incluindo GNU/Linux, Unix (Mac OS X, Solaris), e MS Windows (POSTGRESQL, 2011).

É totalmente compatível com ACID, tem suporte completo a chaves estrangeiras, junções (*joins*), visões, *triggers* e procedimentos armazenados (em múltiplas linguagens). Inclui a maior parte dos tipos de dados do ISO SQL:1999, incluindo *INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL*, e *TIMESTAMP*. Suporta também o armazenamento de objetos binários, incluindo figuras, sons ou vídeos. Possui interfaces nativas de programação para C/C++, Java, Net, Perl, Python, Ruby, ODBC, entre outros, e uma excepcional documentação (POSTGRESQL, 2011).

Como um banco de dados de nível corporativo, o PostgreSQL possui funcionalidades sofisticadas como o controle de concorrência multiversionado

(MVCC), recuperação em um ponto no tempo (PITR), *tablespaces*, replicação assíncrona, transações agrupadas (*savepoints*), *backup online*, um sofisticado planejador de consultas e registrador de transações sequencial (WAL) para tolerância a falhas (POSTGRESQL, 2011).

É altamente escalável, tanto na enorme quantidade de dados que pode gerenciar, quanto no número de usuários concorrentes que pode acomodar. Existem sistemas ativos com o PostgreSQL em ambiente de produção que gerenciam mais de 4TB de dados (POSTGRESQL, 2011). O Quadro 1 mostra alguns dos limites do PostgreSQL.

Limite	Valor
Tamanho máximo do banco de dados	Ilimitado
Tamanho máximo de uma tabela	32 TB
Tamanho máximo de uma linha	1.6 TB
Tamanho máximo de um campo	1 GB
Máximo de linhas por tabela	Ilimitado
Máximo de colunas por tabela	250-1600 dependendo do tipo de coluna
Máximo de índices por tabela	Ilimitado

Quadro 1 – Limites do PostgreSQL
Fonte: Adaptado de: POSTGRESQL (2011)

O PostgreSQL utiliza o modelo de arquitetura cliente-servidor. Uma sessão do PostgreSQL consiste em dois processos:

- Um processo servidor, que é responsável por gerenciar os arquivos de banco de dados. Ele recebe conexões dos aplicativos cliente com o banco de dados e executa ações no mesmo em nome dos clientes. O programa servidor de banco de dados se chama *postmaster;*
- O outro processo é o aplicativo cliente do usuário (*frontend*) que deseja executar operações no banco de dados. Os aplicativos cliente podem ser dos mais variados tipos: um aplicativo gráfico, um servidor *WEB* que acessa o banco de dados para mostrar resultados nas páginas *WEB*, ou então uma ferramenta especializada para manutenção do banco de dados. Alguns

desses aplicativos cliente são fornecidos na distribuição do PostgreSQL, porém a grande maioria é desenvolvido pelos usuários (DOCUMENTAÇÃO DO POSTGRESQL 8.0, 2005).

O servidor PostgreSQL pode tratar várias conexões simultâneas de clientes. Para esta finalidade é iniciado um novo processo (*fork*) para cada conexão. A partir desse momento, o cliente e o novo processo servidor se comunicam sem intervenção do processo *postmaster* original. Portanto, o *postmaster* está sempre executando aguardando por novas conexões dos clientes, enquanto os clientes e seus processos servidores associados surgem e desaparecem (DOCUMENTAÇÃO DO POSTGRESQL 8.0, 2005). A Figura 2 mostra duas conexões simultâneas de clientes no PostgreSQL.

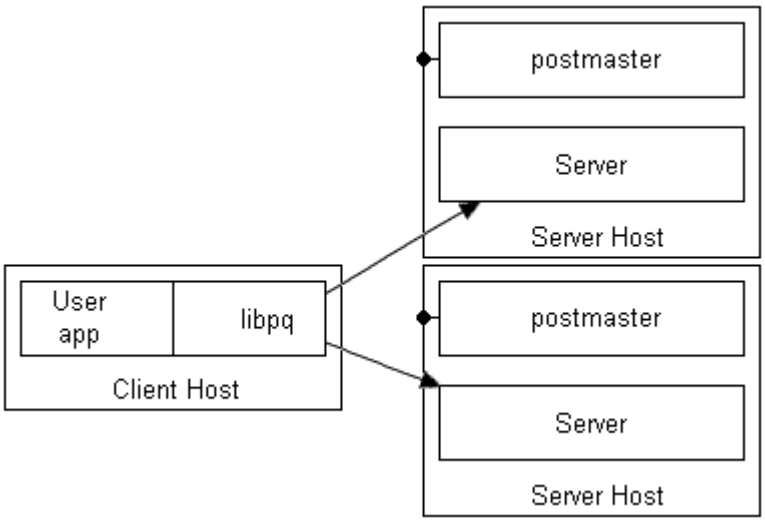

Figura 2 – Múltiplas conexões no PostgreSQL
Fonte: PostgreSQL *Architectural Concepts*

2.5 SERVIDORES DE APLICAÇÃO

Os servidores de aplicação são softwares responsáveis por fornecer toda uma infraestrutura de serviços para o desenvolvimento e execução de uma aplicação distribuída. Uma das vantagens que eles possuem em relação ao modelo cliente-servidor é de oferecer serviços implementados por eles e que estão disponíveis aos desenvolvedores, fazendo com que esses últimos dediquem-se mais no

desenvolvimento da lógica de negócio do que com aspectos de infraestrutura (APPLICATION SERVERS, 2003).

De um modo geral esses serviços diminuem a complexidade do desenvolvimento, controlam o fluxo de dados, aprimoram o desempenho e gerenciam a segurança. Os servidores de aplicações predispõem a utilização da arquitetura conhecida como n-camadas (cliente, servidor de aplicação e banco de dados) permitindo aproveitar melhor as características de cada componente. O *Front-End* é a primeira camada e geralmente são *browsers*, que servem para apresentação da aplicação e fazem algumas validações. A segunda camada é a aplicação em si, que é executada no servidor de aplicação e a terceira é o servidor de banco de dados (APPLICATION SERVERS, 2003). A Figura 3 exemplifica a arquitetura de um servidor de aplicação.

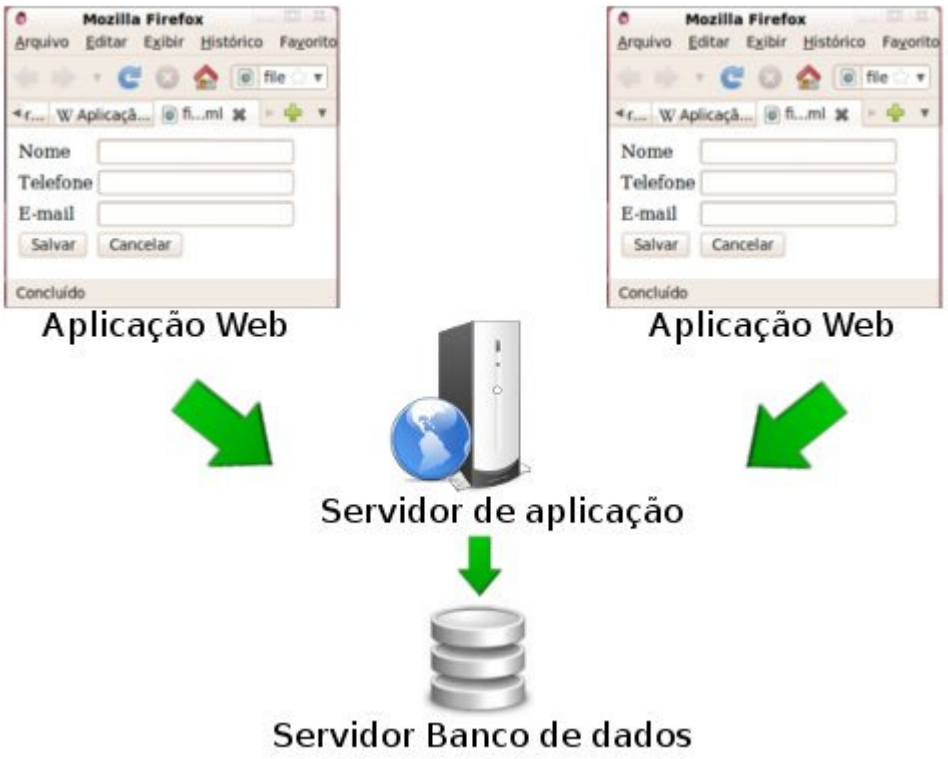

Figura 3 – Arquitetura de um servidor de aplicação
Fonte: JavaEE: Conceitos Básicos (2010)

GlassFish é um servidor de aplicações baseado na Plataforma Java e tecnologia *Enterprise Edition* (Java EE) para o desenvolvimento e execução de aplicações e serviços *WEB*, oferecendo desempenho, confiabilidade, produtividade e

facilidade(GLASSFISH, 2009). É um servidor de aplicação de código aberto (*open source*) disponível gratuitamente e está licenciado sob *Common Development and Distribution License* (CDDL) (HEFFELFINGER, 2010).

Atualmente o GlassFish se encontra na versão 3.1 sendo o primeiro servidor de aplicação a dar suporte total a especificação Java EE 6. A sua nova versão conta com funcionalidades que adicionam características como alta disponibilidade, escalabilidade e tolerância a falhas, que são fundamentais para um servidor de produção que hospede aplicações de médio e grande porte (SOUZA, 2011).

Além das características citadas a versão 3.1 conta com outra serie de características e funcionalidades:

- Evocação dinâmica de serviços (somente os serviços utilizados são iniciados);
- Melhorias de extensibilidade, tanto no servidor quanto na interface de administração;
- Ciclos de inicialização e implantação (*deploy*) 29% mais rápidos que na versão 3.0.1;
- Sessão, *Stateful Session Beans* e EJB *Timers* são preservados entre *re-deploys*;
- Melhor integração com ferramentas (NetBeans, Eclipse, plug-in Maven para testes unitários);
- Desempenho da funcionalidade de alta disponibilidade 34% melhor do que a versão 2.1.1;
- Administração centralizada dos *clusters*, com escalabilidade melhorada (até 100 instâncias gerenciadas);
- Suporte a autenticação via PAM;
- Suporte a *upgrade* da versão 2.x ou 3.x para a 3.1;
- Mais funcionalidades disponíveis na versão comercial do GlassFish (Oracle GlassFish *Server*), como interoperabilidade com WEBSphere e WEBLogic, *backup* e recuperação, analisador de desempenho, balanceamento de carga, etc (SOUZA, 2011).

O GlassFish ainda fornece suporte para a criação e administração de *realms.* Um *realm* é um repositório de usuários e grupos que possui mecanismos para

identificar usuários válidos para uma aplicação (JAVA EE 5 TUTORIAL, 2010). É importante definir alguns conceitos que compõe um *realm:*

- Usuário (*User*): um usuário é um indivíduo com uma identidade que foi definida no servidor de aplicação. Em uma aplicação *WEB*, um usuário pode ter um conjunto de papéis (*roles*) associados a essa identidade, que lhes permite acessar todos os recursos protegidos por essas funções. Os usuários podem ser associados com um grupo de usuários;
- Grupo (*Group*): um grupo pode ser definido como um grupo de usuários que possuem características em comum;
- Pápeis (*Roles*): Uma *Role* pode ser definida como um conjunto de privilégios que pode ser associado a um determinado grupo de usuários. Uma *role* pode ser comparada a uma chave que pode abrir um cadeado. Muitas pessoas podem ter uma cópia da chave e para a fechadura não importa quem a pessoa é, só interessa que ela tenha a chave certa (JAVA EE 5 TUTORIAL, 2010).

3 ESTUDO EXPERIMENTAL

A aplicação *WEB* desenvolvida para este trabalho se baseia em uma agenda *on-line,* onde uma pessoa poderia cadastrar seu estabelecimento (Exemplo: uma clínica de dentistas). Também seria possível cadastrar os funcionários desse estabelecimento e cada funcionário teria sua agenda onde seriam marcados os horários das suas consultas.

3.1 MATERIAIS E MÉTODOS

Para o desenvolvimento deste projeto, foi realizada uma ampla pesquisa bibliográfica, além de diversas consultas em sites especializados no assunto sobre segurança e desempenho em aplicações *WEB* , a fim de dar subsídios suficientes à implementação das tecnologias em estudo.

Para demonstrar o funcionamento e as configurações dos mecanismos de segurança e desempenho que vão ser descritos nos próximos capítulos foi desenvolvida uma aplicação *WEB* na plataforma Java EE 6, utilizando também alguns recursos do PrimeFaces 2.2, que é uma biblioteca de componentes para JSF 2.0. Foi utilizado também o mecanismo de autenticação e autorização do Java (JAAS). O servidor de aplicação utilizado foi o GlassFish *Server Open Source Edition 3* e o banco de dados foi o PostgreSQL 9.0. Também foram utilizadas algumas funcionalidades do sistema operacional *Linux*.

3.2 REQUISITOS DE SEGURANÇA DA APLICAÇÃO

A aplicação *WEB* desenvolvida tem a necessidade de fornecer os seguintes requisitos de segurança e desempenho:

- Impedir que usuários não cadastrados ou não autorizados acessem páginas ou recursos restritos;

- Fazer com que o usuário tenha que efetuar *login* novamente no sistema caso a sessão fique mais de 20 minutos inativa;
- Fazer a validação de cadastro por *email* para impedir que sejam cadastrados usuários com *emails* que não existem;
- Implementar um mecanismo para impedir que programas automáticos não consigam realizar cadastros e com isso possam sobrecarregar a aplicação;
- Otimizar o desempenho da aplicação para aumentar o nível de satisfação do usuário;
- Realizar *backups* automaticamente para que essa tarefa não precise ser delegada a uma pessoa.

3.3 CONFIGURAÇÃO DO JAAS

O primeiro mecanismo de segurança a ser implantado será o JAAS. Ele atenderá o primeiro requisito de segurança, que é impedir que usuários não cadastrados ou não autorizados acessem páginas ou recursos restritos. As configurações para a implementação do JAAS acontecem em três lugares: no banco de dados, nas configurações do GlassFish e na aplicação em si.

3.3.1 Esquema do Banco de Dados

A criação das tabelas que serão usadas para definir os usuários e os grupos de usuários faz parte do primeiro passo para a configuração do JAAS.

A tabela `estabelecimento` será a tabela onde estarão guardados os dados dos usuários da aplicação, tendo como atributos fundamentais para a configuração do JAAS os campos `login` e `senha`.

A tabela `groups` tem apenas dois campos: o campo `idgroup` onde serão inseridos os grupos de usuários que serão definidos na aplicação (`admins` e `users`) e o campo `login` que é uma chave estrangeira do atributo `login` da tabela

`estabelecimento`. Essa tabela vai ter uma chave composta formada por esses dois campos (`idgroup` e `login`) para que um usuário não seja associado duas vezes a um mesmo grupo.

É importante ressaltar que as regras de acesso (*roles*) que serão feitas na aplicação vão ser aplicadas aos grupos de usuários e não aos usuários em si. A Figura 4 apresenta o relacionamento entre as tabelas `estabelecimento` e `groups`.

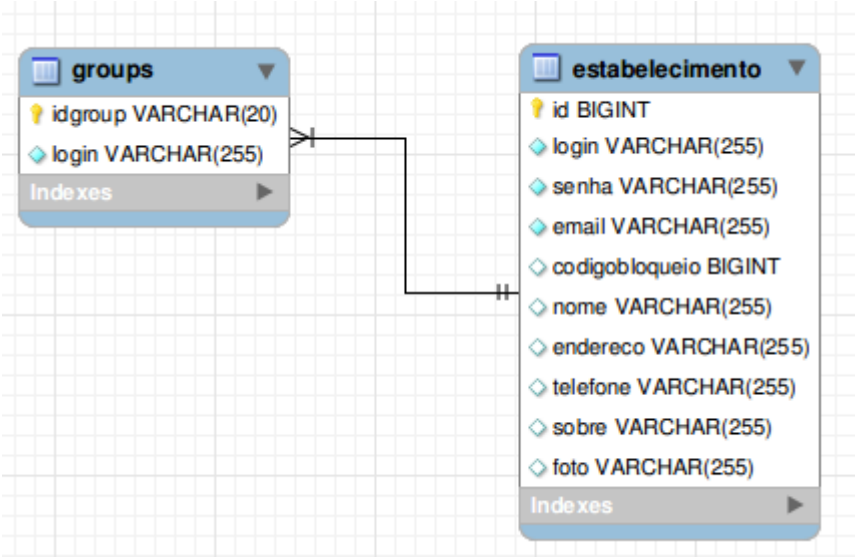

Figura 4 – Relacionamento entre as tabelas estabelecimento e *groups*

3.3.2 Configurando o JAAS no GlassFish

Com o banco de dados criado e configurado é necessário criar um *pool* de conexão (*connection pool*) e um *data source* que fará a conexão com esse *database.* Essas configurações são feitas no painel de administração do GlassFish que pode ser acessado em qualquer navegador *WEB* através do endereço *localhost:4848*.

A criação de *connection pools* é possível através do recurso `JDBC Connection Pools` e um exemplo de configuração pode ser verificado na Figura 5. Ainda na configuração do *connection pool,* deve-se informe as propriedades

especificas do banco de dados, como o nome, a senha e o usuário do *database.* As demais propriedades podem ser as padrões.

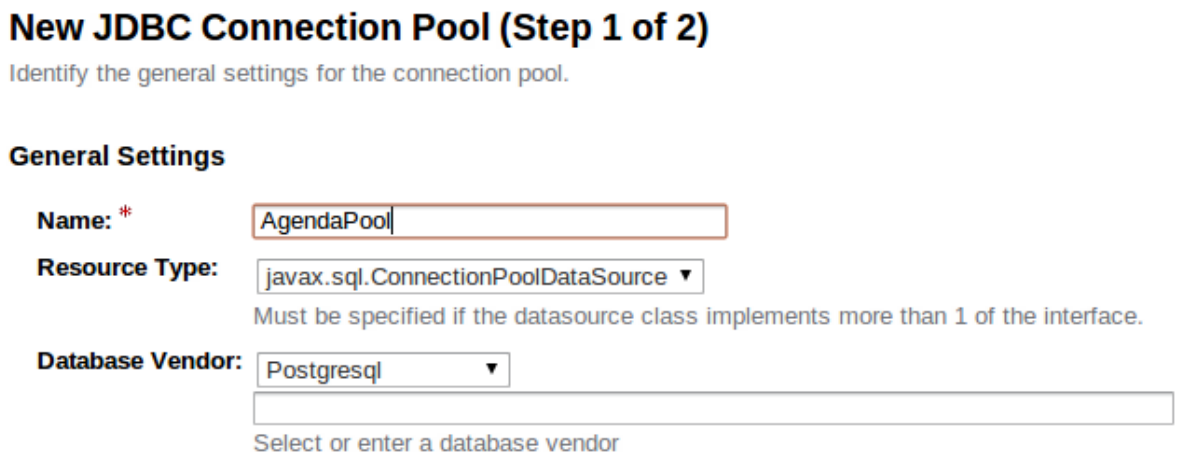

Figura 5 – Configuração de um *conection pool*

Agora é necessário criar o *data source* que irá utilizar o *connection pool* que foi feito anteriormente. Essa configuração é possível através do recurso `JDBC Resources`. A Figura 6 mostra um exemplo das propriedades de um novo *data source.*

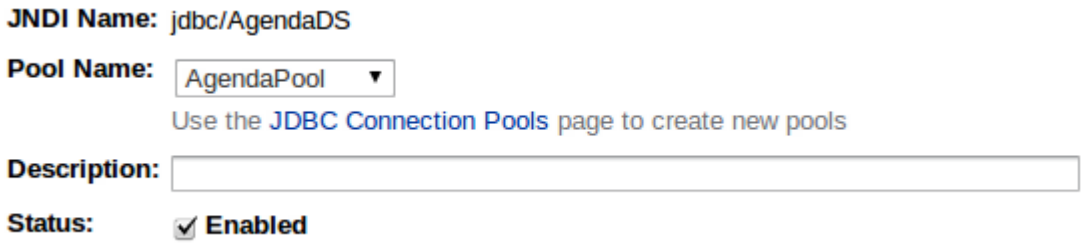

Figura 6 – Configuração de um *data source*

Depois de criado o *data source* podemos configurar o *jdbcRealm* que será usado na aplicação. O GlassFish fornece o recurso de criação de *realms,* sendo possível visualizar no Quadro 2 algumas propriedades importantes na hora de criação de um novo *realm.*

Propriedade	Valor
Realm Name	agenda-jaas-realm
Class Name	com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm
JAAS Context	jdbcRealm
JNDI	jdbc/AgendaDS
User Table	estabelecimento
User Name Column	login
Password Column	senha
Group Table	groups
Group Name Column	idgroup
Database User	postgres
Database Password	postgres
Digest Algorithm	md5

Quadro 2 - Configuração de um novo *realm*

É importante que o algoritmo de criptografia (*Digest Algorithm*) configurado no *realm,* seja o mesmo que será usado para criptografar a senha do usuário que irá ser inserida no banco de dados. O Quadro 3 mostra o método criado na aplicação que é encarregado por criptografar em `md5` a senha do usuário na hora do cadastro. MD5 é um algoritmo de *hash* unidirecional e, portanto, uma vez criptografados, os dados não podem mais ser descriptografados.

```
1 public static String toMD5(String senha) {
2     MessageDigest md;
3     String senhaMD5 = senha;
4     try {
5     md = MessageDigest.getInstance("MD5");
6     BigInteger hash = new BigInteger(1,
7                         md.digest(senha.getBytes("UTF-   7 8")));
8     senhaMD5 = hash.toString(16);
9           while (senhaMD5.length() < 32)
10          {
11                      senhaMD5 = "0" + senhaMD5;
12          }
13    }
14    catch (NoSuchAlgorithmException e) {} catch
15          (UnsupportedEncodingException e) {}
16    return senhaMD5; }
```

Quadro 3 - Método para criptografar senha em md5

3.3.3 Configurando o JAAS na Aplicação

A configuração na aplicação é a ultima parte da implantação do mecanismo de segurança JAAS. Essa configuração é feita basicamente em dois arquivos da aplicação *WEB:* no `web.xml` e no `sun-web.xml`.

Primeiramente será criado as *roles* que farão parte da aplicação. Essas *roles* serão associadas aos grupos de usuários e não diretamente aos usuários. Um ponto importante é que uma *role* pode estar associada a mais de um grupo de usuário.

O Quadro 4 mostra um exemplo de como criar roles em um arquivo `sun-web.xml`

```
1 <security-role-mapping>
2     <role-name>user-role</role-name>
3     <group-name>admins</group-name>
4     <group-name>users</group-name>
5 </security-role-mapping>
6
7 <security-role-mapping>
8     <role-name>admin-role</role-name>
9     <group-name>admins</group-name>
10 </security-role-mapping>
```

Quadro 4 - Criação de *roles* em um arquivo `sun-web.xml`

Como pode ser visto no Quadro 4, o grupo `admins` além de estar associado a *role* `admin-role` está também na `user-role`. Isso foi feito para que o grupo `admins` tenha acesso a todas as páginas, caso contrário ele seria impedido de acessar uma página que está controlada pela *role* `user-role`. Esses grupos serão os mesmos que estarão na tabela `groups` no banco de dados. Para fazer essa associação, no momento do cadastro de um novo usuário, deve definir qual será seu grupo (`admins` ou `users`) e gravá-lo na tabela `groups`.

A tarefa seguinte consiste na configuração do arquivo `web.xml`. É nele onde dizemos qual será o método de autenticação (*none, basic, form* ou *digest/certificate*) e qual o nome do *realm* associado. O Quadro 5 mostra a configuração de um `login-config` e de uma `error-page`.

```
1 <login-config>
2    <auth-method>FORM</auth-method>
3    <realm-name>agenda-jaas-realm</realm-name>
4    <form-login-config>
5     <form-login-page>/views/login/login-jaas.jsf</form-login-page>
6     <form-error-page>/views/login/login-error-jaas.jsf</form-error-page>
7    </form-login-config>
8 </login-config>
9
10 <error-page>
11    <error-code>403</error-code>
12    <location>/views/login/permission-error.jsf</location>
13 </error-page>
```

Quadro 5 - Configuração de um *login-config* e de uma *error-page*

No exemplo do Quadro 5, o método de autenticação utilizado foi *FORM*, ou seja, é necessário informar qual vai ser a página de *login* e a página de erro de *login*, caso o usuário informe dados incorretos. Também foi configurada uma página de erro para o código 403, que será disparado caso um usuário seja autenticado porém não esteja autorizado a acessar um determinado recurso.

Ainda no arquivo `web.xml` existe uma configuração importante, que é utilizar as *roles* criadas no arquivo `sun-web.xml` para definir as regras de acesso na aplicação. O Quadro 6 mostra o exemplo da configuração de uma regra de acesso.

```
1 <security-constraint>
2     <web-resource-collection>
3          <web-resource-name>agenda do estabelecimento</web-resource-name>
4          <url-pattern>/views/agenda/*</url-pattern>
5          <http-method>GET</http-method>
6          <http-method>POST</http-method>
7     </web-resource-collection>
8     <auth-constraint>
9           <role-name>user-role</role-name>
10    </auth-constraint>
11 </security-constraint>
```

Quadro 6 - Configuração de uma regra de acesso

Como pode ser visto no Quadro 6 foi criado um recurso onde está definido que todas as *urls* a partir do caminho `/views/agenda/` serão restritas a *role* `user-role`. É possível criar quando regras forem necessárias, tudo depende da necessidade de controle que a aplicação precisa ter.

Por fim, deve-se criar a página de *login*. Essa página é composta por um formulário simples. Um detalhe importante e obrigatório é colocar os nomes do campo `login` e `senha` como `j_username` e `j_password` e na `action` do `form`

colocar `j_security_check` para que o servidor de aplicação identifique que se trata de uma tentativa de *login* e faça a autenticação necessária. O Quadro 7 mostra o exemplo de um formulário de *login* com JAAS.

```
 <form method="post" action="j_security_check" class="login">
   <h3>Digite seu login e senha:</h3>
   <input type="text" id="username" name="j_username" value="usuario" />
   <input type="password" id="password" name="j_password" value="senha" />
   <input type="submit" name="submit" value="Login" />
</form>
```

Quadro 7 – Formulário de *login* com JAAS

Após um usuário ser autenticado pelo servidor de aplicação ele irá ter suas credenciais guardadas em uma sessão e não precisará se autenticar novamente até que a sessão termine ou expire.

3.4 OUTROS MECANISMOS DE SEGURANÇA

O JAAS é o principal mecanismo de segurança que foi implementado nesse estudo experimental. Entretanto, ele não atende a necessidade de todos os requisitos de segurança que foram especificados. Para atender essas necessidades foram implementados outros mecanismos mais simples, porém eficazes.

3.4.1 Phase Listener

Para atender ao requisito que especifica que um usuário não deve ficar mais de 20 minutos inativo, foi implementando um *Phase Listener*.

O *Phase Listener* é um recurso especifico do JSF que é responsável por interceptar e oferecer mecanismos de manipulações referentes às mudanças de eventos ocorridas no ciclo de vida da especificação (FRANZINI, 2009).

Para utilizar esse recurso, primeiro é necessário criar uma classe que implemente a *interface PhaseListener*. Essa *interface* possui três métodos:

`afterPhase()`, `beforePhase()` e `getPhaseId()`. No Quadro 8 é possível ver um exemplo da implementação do *Phase Listener.*

```
1 public class SessionListener implements PhaseListener {
2
3     private static final long serialVersionUID = 1L;
4
5     public void afterPhase(PhaseEvent event) {
6         FacesContext facesContext = event.getFacesContext();
7         HttpSession session = (HttpSession)
8                 facesContext.getExternalContext().getSession(false);
9
10        if(facesContext.getExternalContext().getUserPrincipal() != null)
11        {
12          long lastAcessedTime = Calendar.getInstance().getTimeInMillis()
13           - session.getLastAccessedTime();
14      lastAcessedTime = TimeUnit.MILLISECONDS.toMinutes(lastAcessedTime);
15
16                if(lastAcessedTime > 20)
17                {
18                      session.invalidate();
19                      String loginPage = "/views/login/login-jaas.jsf";
20                      ExternalContext externalContext =
21                              facesContext.getExternalContext();
22                      try {
23    externalContext.redirect(externalContext.getRequestContextPath()
24                         +loginPage);
25                      } catch (IOException e) {
26                           e.printStackTrace();
27                      }
28                }
29          }
30    }
31
32    public void beforePhase(PhaseEvent event) {}
33          public PhaseId getPhaseId() {
34          return PhaseId.RESTORE_VIEW;
35    }
36 }
```

Quadro 8 – Implementação da interface *PhaseListener*

Na classe mostrada no Quadro 8**,** a lógica de negócio para controlar o tempo de sessão do usuário é implementado no método `afterPhase()`, onde será feita uma validação para verificar se existe um usuário na sessão. Caso o usuário exista, será verificado o tempo que ele está inativo (não fez nenhuma requisição na aplicação) e se ele estiver por mais de 20 minutos inativo a sessão será terminada e o usuário será redirecionado para a página de *login*. A classe `SessionListener` será chamada na primeira fase do ciclo de vida do JSF, chamada de *Restore View*.

Depois de criada, essa classe deve ser configurada no arquivo `faces-config.xml` para que a aplicação saiba que se trata de uma implementação da *interface PhaseListener*. O Quadro 9 mostra um exemplo dessa configuração.

```
1 <lifecycle>
2     <phase-listener>agenda.utils.SessionListener</phase-listener>
3 </lifecycle>
```

Quadro 9 – Configuração da classe `SessionListener` no arquivo `faces-config.xml`

3.4.2 Validação de cadastro por email

Fazer uma validação de novos cadastros por *email* é uma maneira eficiente para evitar que pessoas de má fé se registrem inúmeras vezes com *emails* que não existem, deixando o banco de dados com informações inconsistentes e que não poderão ser utilizadas.

Para implementar essa validação por *email*, primeiro é necessário configurar o servidor de aplicação para que ele possa enviar *emails*. No painel de administração do GlassFish é possível criar um recurso chamado de *JavaMail Session*, que implementa essa funcionalidade. O Quadro 10 mostra as propriedades de uma *JavaMail Session* que, nesse caso é configurada levando em conta as propriedades do serviço de *email* do *gmaill*.

Propriedade	**Valor**
JNDI Name:	email/agendadahora@gmail.com
Mail Host:	smtp.gmail.com
Default User	agendadahora@gmail.com
Default Return Address	agendadahora@gmail.com
mail.smtp.socketFactory.port	465
mail.smtp.port	465
mail.smtp.socketFactory.fallback	false
mail.smtp.auth	true
mail.smtp.password	lejafetcc
mail.smtp.socketFactory.class	javax.net.ssl.SSLSocketFactory

Quadro 10 – Propriedades de uma *JavaMail Session*
Fonte: Adaptado de: *Using JavaMail API with Glassfish and Gmail* (2010)

Depois de configurado esse serviço no servidor de aplicação é necessário criar uma classe EJB na aplicação, que será responsável por instanciar o recurso de *email* criado no GlassFish e mandar o *email* para o usuário. O Quadro 11 mostra o exemplo da classe que é responsável pelo envio de *emails*.

```
1 @Stateless
2 @Remote(EmailEJBRemote.class)
3 public class EmailEJB implements EmailEJBRemote, Serializable{
4
5     private static final long serialVersionUID = 1L;
6
7     @Resource(name="email/agendadahora@gmail.com")
8     private Session mailSession;
9
10    @Override
11    public void sendMessage(String email, String assunto, String
12    mensagem) {
13          Message msg = new MimeMessage(mailSession);
14          try{
15                msg.setSubject(assunto);
16                msg.setRecipient(RecipientType.TO, new
17                          InternetAddress(email));
18                msg.setText(mensagem);
19                Transport.send(msg);
20
21          }catch(MessagingException me){
22                me.printStackTrace();
23          }
24    }
25 }
```

Quadro 11 – Classe responsável por fazer o envio de *emails*

A entidade `estabelecimento`, que representa o cadastro do usuário, possui um atributo chamado `codigoBloqueio`. O código de bloqueio é criado no momento em que o usuário efetua o cadastro, através do método mostrado no Quadro 12.

```
1 public Long gerarNumeroDesbloqueio(){
2     String numero = "";
3     for(int i=0;i<8;i++){
4           numero+=new Long(Math.abs(new
5           Random().nextLong())).toString().charAt(0);
6     }
7     return new Long(numero);
8 }
```

Quadro 12 – Método responsável por gerar o código de bloqueio do cadastro

Após realizar o cadastro o usuário receberá um *email* com um *link* que contém como parâmetros o código de bloqueio e o *email* do próprio usuário. Acessando esse *link* o usuário será direcionado para a página de desbloqueio que receberá os parâmetros que foram passados na *url*. Com isso, basta o usuário avançar com o cadastro que terá a sua conta desbloqueada. O método que efetua o desbloqueio do cadastro é mostrado no Quadro 13.

```
1 @Override
2 public boolean desbloquearEstabelecimento(Long codigoDeDesbloqueio,
3                                     String email) throws Exception {
4
5     //Faz uma busca pelo estabelecimento através do email
6    Query query =
7    em.createNamedQuery("Estabelecimento.buscarEstabelecimentoPorEmail");
8    query.setParameter("email", email);
9
10    @SuppressWarnings("unchecked")
11    List<Estabelecimento> lista = query.getResultList();
12    if(lista==null || lista.isEmpty()){
13          return false;
14    }
15    Estabelecimento estabelecimento = lista.get(0);
16
17    //Verifica se o codigo do desbloqueio confere
18
19    if(estabelecimento.getCodigoBloqueio().equals(codigoDeDesbloqueio)){
20          //Desbloqueia o estabelecimento
21          estabelecimento.desbloquearEstabelecimento();
22          this.salvar(estabelecimento);
23          return true;
24    }
25    return false;
26 }
```

Quadro 13 – Método responsável por realizar o desbloqueio do cadastro

3.4.3 Captcha

Um outro requisito de segurança importante é impedir que programas automatizados não consigam realizar cadastros no sistema. Para impedir esse tipo de ação existe um mecanismo simples, porém muito eficaz que é conhecido como Captcha.

Captcha é um acrônimo para *Completely Automated Public Turing Test to Tell Computers and Humans Apart*, ou seja, um teste público totalmente automatizado para diferenciar humanos de computadores (LANDIM, 2009).

De um modo geral, o Captcha serve como um recurso auxiliar para evitar *spams* ou mensagens disparadas por outros computadores ou robôs. A idéia é que as respostas dos testes que são feitos no Captcha sejam impossíveis de serem respondidas por um computador, permitindo assim que somente seres humanos tenham acesso a determinados conteúdos ou possam enviar informações (LANDIM, 2009).

O Captcha utilizado na aplicação é fornecido pela biblioteca de componentes do Prime Faces. Esse Captcha, além de fazer um teste simples onde duas palavras mostradas devem ser digitadas corretamente, dá a opção para o usuário poder ouvir um outro teste que é feito especialmente para pessoas com deficiência visual. Isso é algo muito importante, pois a aplicação precisar ser acessível para todos.

Para implementar o Captcha na aplicação é necessário primeiramente declarar o componente em uma página `.xhtml`. Essa ação é exemplificada no Quadro 14.

```
1 <!-- Captcha -->
2 <h:panelGroup id="captchaGroup" styleClass="groups">
3   <h:message for="captcha" styleClass="error-       message" />
4   <br />
5    <span>
6  <p:captcha id="captcha" label="Captcha" language="pt" theme="white" />
7    </span>
8  </h:panelGroup>
```

Quadro 14 – Declarando o componente Captcha em uma página `.xhtml`

Para o Captcha funcionar corretamente, existe também a necessidade de criar uma chave pública e privada para esse componente. Para isso, é necessário que seja feito um cadastro no serviço do Google chamado ReCaptcha. Com esse cadastro, será gerada uma chave pública e uma chave privada que serão utilizadas pelo sistema para garantir a segurança do mesmo. O Quadro 15 mostra como configurar essas chaves na aplicação dentro do arquivo `web.xml`.

```
1 <context-param>
2     <param-name>primefaces.PUBLIC_CAPTCHA_KEY</param-name>
3     <param-value>6LcNIcISAAAAAH4uIL6BCSPCNxdJV189t6a2ia2-</param-value>
4 </context-param>
5 <context-param>
6     <param-name>primefaces.PRIVATE_CAPTCHA_KEY</param-name>
7     <param-value>6LcNIcISAAAAAA7kF4imF6QW3urfwqVURwJF8fy3</param-value>
8 </context-param>
```

Quadro 15 – Configuração das chaves pública e privada do Captcha

Com essas configurações a página de cadastro da aplicação estará protegida contra ataques de programas automatizados. A Figura 7 mostra a página de cadastro com o Captcha implementado.

Figura 7 – Captcha na página de cadastro

3.5 OTIMIZANDO O DESEMPENHO DA APLICAÇÃO

Não basta apenas deixar a aplicação segura. É necessário também configurar mecanismos que otimizem o seu desempenho para melhorar a satisfação do usuário e ter uma maior aceitação perante os mesmos. Existem algumas funcionalidades,

que podem ser configuradas no GlassFish que aumentam consideravelmente a performance da aplicação.

3.5.1 Compressão GZIP

Gzip é a abreviação de GNU zip, um Software Livre de compressão sem perda de dados. A compressão por gzip é uma maneira bastante eficiente para otimizar o desempenho de um site. Consiste em enviar os códigos do site em formato comprimido, para que ocupem menos espaço e com isso trafeguem pela rede de forma mais rápida (ANTUNES, 2010).

Para implementar esse protocolo os sistemas cliente e servidor trocam informações de controle entre si, nos cabeçalhos HTTP da solicitação e a das respostas. Primeiro o servidor verifica se o navegador tem suporte à compressão e caso tenha, o servidor irá comprimir os arquivos e então mandá-los para o *browser*. Caso o navegador não tenha suporte à compressão gzip os arquivos serão enviados normalmente sem serem comprimidos. Entretanto, atualmente todos os navegadores dão suporte à essa compressão (ANTUNES, 2010).

O GlassFish permite ativar a compressão gzip mudando algumas configurações nos *Network Listeners.* A Figura 8 mostra as configurações feitas para ativar a compressão gzip. Essas configurações foram feitas no `http-listener-1` que usa como porta padrão a 8080.

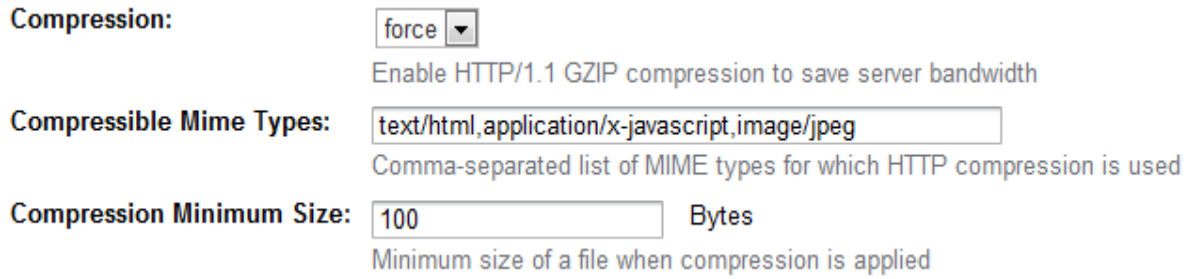

Figura 8 – Ativação da compressão gzip no GlassFish

No campo *Compressible Mime Types* são informados os tipos de arquivos que serão comprimidos pelo servidor. O Quadro 16 mostra uma lista com os tipos mais comuns de *mime types* e a sua respectiva extensão.

Extensão	Mime Type
.html	text/html
.xml	text/xml
.js	application/x-javascript
.jpeg	image/jpeg
.jpg	image/jpg
.bmp	image/bmp
.gif	image/gif
.png	image/png
.avi	video/avi
.css	text/css

Quadro 16 – Principais tipos de *mime type*
Fonte: Adaptado de: Webmaster Toolkit (2011)

3.5.2 Habilitando o cache de arquivos

Outro recurso que pode ser habilitado no `http-listener-1` é a ativação de *cache* para os arquivos da aplicação. Com isso os arquivos vão ser guardados no *cache* do navegador e serão carregados mais rapidamente, aumentando o desempenho da aplicação. A Figura 9 mostra as configurações para a ativação desse recurso.

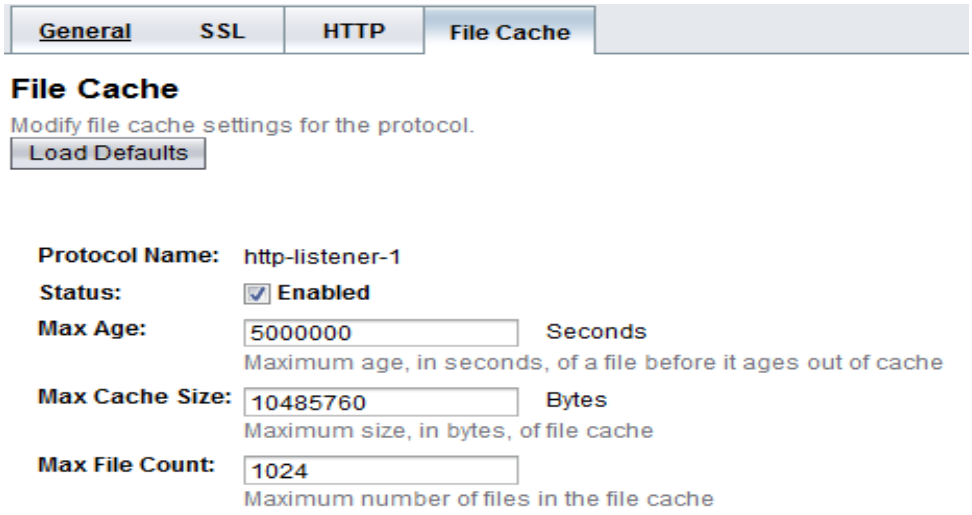

Figura 9 – Habilitando o *cache* de arquivos

Para testar o impacto real dessas modificações na aplicação, foi utilizado o *plugin Page Speed* para o navegador Google Chrome. A Figura 10 mostra o resultado do teste do *Page Speed* na página inicial da aplicação antes de ativar os recursos de compressão e *cache*. É possível perceber que ele destaca em vermelho e amarelo os itens de `Ativar compactação` e `Aproveitar cache do navegador`.

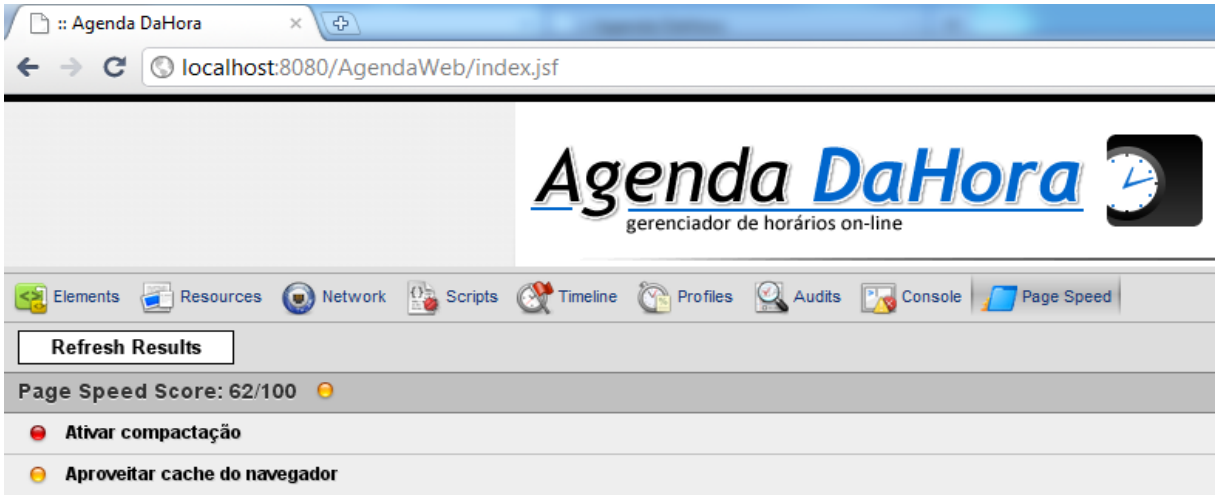

Figura 10 – Teste do *Page Speed* sem os recursos de compressão e *cache*

Realizando outro teste do *Page Speed*, agora com os recursos de compressão gzip e *cache* ativos, é possível perceber que houve um aumento significativo no desempenho da aplicação. A Figura 11 mostra o resultado desse teste.

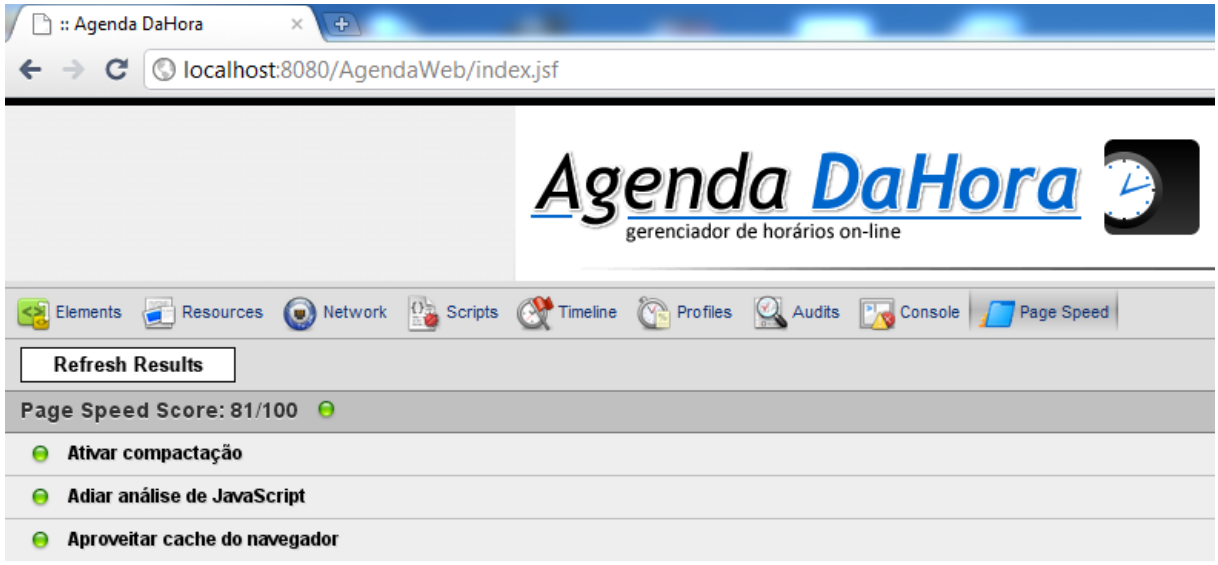

Figura 11 – Teste do *Page Speed* com os recursos de compressão e *cache* ativos

3.6 AUTOMATIZAÇÃO DE BACKUPS

Realizar *backups* periodicamente é uma tarefa essencial para manter a segurança e integridade dos dados do sistema. Entretanto, realizar essa tarefa manualmente pode se tornar algo trabalhoso e pode acontecer da pessoa encarregada por realizar os *backups* acabar se esquecendo de realizar tal tarefa.

Para que esse problema não aconteça, existe a possibilidade de deixar que ferramentas do sistema operacional onde está instalado o SGBD, sejam responsáveis por realizar os *backups*. Nesse estudo experimental a automatização de *backups* foi configurada no sistema operacional Linux.

Inicialmente é necessário criar um *script shell,* que será responsável por realizar o *dump* da base de dados. O Quadro 17 mostra o *script* que realizará tal tarefa.

```
#!/bin/bash

# Define as datas
dia=`/bin/date +%d-%m-%Y`
diaHora=`/bin/date +%d%m%Y%H%M`

# Define o diretorio principal dos backups
diretorio="/home/jariel/Documents/backups/"

# Cria o diretório do dia se ele não existir
if [ -d $diretorio/$dia ]; then
cd $diretorio/$dia
else
`mkdir $diretorio/$dia`
fi

# Faz o dump dos dados para o diretorio do dia, criando um arquivo com dia e hora
pg_dump -Ft --username postgres --blobs --verbose --file "$diretorio/$dia/$diaHora"-agendaDB.tar"" agendaDB
```

Quadro 17 – *Script shell* responsável por realizar o *dump* da base de dados

O principal comando realizado nesse *script* é o `pg_dump`. O `pg_dump` é um comando do PostgreSQL que tem como função fazer o *dump* dos dados. Nele informamos parâmetros como usuário e o nome do banco de dados, e também o diretório onde será salvo o *backup*. Para manter uma organização sobre os *backups*, o *script* criará um diretório com a data atual, onde será salvo o arquivo.

Com o *script* pronto é necessário agendá-lo para ser executado diariamente. Para tal ação é utilizado o cron. O cron é um programa que executa comandos agendados, nos sistemas operacionais do tipo Unix. Para adicionar uma nova tarefa no cron, no terminal do Linux digite o comando `crontab -e`. A Figura 12 mostra o

arquivo `crontab` aberto, que já tem adicionado o comando que será responsável por executar o *script* criado anteriormente.

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

59 23 * * * /home/jariel/Documents/backups/make-backup-agenda.sh
```

Figura 12 – Arquivo `crontab` responsável por agendar tarefas

As ordens dos parâmetros para agendar uma nova tarefa são: *minutos*, *hora*, *dia do mês*, *mês, dia da semana* e o *comando*. Isso quer dizer que nesse caso o *backup* será feito diariamente às 23h59min.

Também foi criado um *script shell* que terá a função de restaurar o banco de dados com base em um arquivo de *backup*. O Quadro 18 mostra os comandos executados nesse *script*, sendo o principal o `pg_restore`, que também é um comando do PostgreSQL e tem como função restaurar a base de dados.

```
#!/bin/bash

# Informe o diretorio que esta o backup
diretorio="diretorio"

# Informe o nome do arquivo de backup
arquivoBKP="backup"

# Delete o banco de dados
dropdb --username postgres agendaDB

# Cria novamente o banco de dados
createdb --username postgres agendaDB

# Restaura o banco de dados com base no backup que foi feito
pg_restore --username postgres --dbname agendaDB $diretorio/$arquivoBKP
```

Quadro 18 – *Script shell* que faz a restauração do banco de dados

4 CONSIDERAÇÕES FINAIS

Esse capítulo tratará das conclusões finais que foram abstraídas durante o desenvolvimento do projeto. Também serão sugeridas algumas idéias para pessoas que leram o trabalho e desejam dar continuidade no mesmo.

4.1 CONCLUSÃO

O desenvolvimento de aplicações *WEB* torna-se cada vez algo mais simples devido ao imenso número de tecnologias e ferramentas disponíveis para desenvolvê-las. Porém deve-se tomar um cuidado especial em pontos como a segurança e o desempenho das mesmas, pois isso se torna um diferencial importante na hora da aprovação do usuário.

O serviço de autenticação e autorização do Java (JAAS) permite ao desenvolvedor de aplicações *WEB* na plataforma J2EE implementar uma mecanismo rápido e eficaz para controlar o acesso dos usuários na aplicação, sem ter que usar nenhum *framework* adicional. Também é possível corrigir outras falhas de segurança utilizando algumas soluções mais simples, como a validação de cadastro por *e-mail,* o controle de tempo de sessão através do *Phase Listener*, e a utilização do Captcha para controlar *spams* e cadastros automatizados.

Ainda no contexto de segurança, porém explorando a área de banco de dados, é possível desenvolver uma solução para a automatização de *backups* no PostgreSQL, utilizando ferramentas do próprio sistema operacional onde está guardada a base de dados.

Por fim, para melhorar o desempenho da aplicação, pode-se utilizar recursos oferecidos pelo servidor de aplicação GlassFish, que possui uma versão disponibilizada gratuitamente.

Aliando todos os mecanismos descritos é possível desenvolver uma aplicação segura e rápida utilizando apenas tecnologias e ferramentas gratuitas.

4.2 TRABALHOS FUTUROS

Como sugestão para trabalhos futuros, fica a idéia de explorar melhor os recursos de segurança oferecido pelo SGBD PostgreSQL, como a definição de usuários e *roles* e suas respectivas permissões, de acesso, edição e alterações de informações.

5 REFERÊNCIAS BIBLIOGRÁFICAS

ANTUNES, Wikerson. **Como funciona a compressão GZIP** / **Delfate**. 2010. Disponível em < http://www.oficinadanet.com.br/artigo/programacao/como_funciona_a_compressao_gzip_deflate >. Acesso em 05 mai. 2011, 15:50.

APPLICATION SERVERS. **Servidores de Aplicações**. 2003. Disponível em < www.iweb.com.br/iweb/pdfs/20031008-appservers-01.pdf >. Acesso em 03 mai. 2011, 23:08.

BROWN, Simon et al. **Pro JSP 2.** 4 ed. Berkeley: Apress, 2005.

CAMP, Jefferson. **Glassfish – Servidor de Aplicações Java**. 2008. Disponível em < http://jeffcamp.wordpress.com/2008/02/03/glassfish-servidor-de-aplicacoes-java/ >. Acesso em 10 mar. 2011, 13:50.

DOCUMENTACAO DO POSTGRESQL 8.0. **Documentação do PostgreSQL 8.0.0**. 2005. Disponível em < http://wiki.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=manual_pg.pdf.zip >. Acesso em 18 abr. 2011, 15:10.

FRANZINI, Fernando. **Autenticação e Autorização em Aplicativos Web**. 2009. Disponível em: < http://imasters.com.br/artigo/14152/java/autenticacao-e-autorizacao-em-aplicativos-web >. Acesso em: 27 abr. 2011, 22:50.

GARTNER. **Segurança das Aplicações Web**. 2004. Disponível em < http://www.risco.org.br/AF/ >. Acesso em 10 mar. 2011, 18:03.

GLASSFISH. **GlassFish - Pesquisas de Compiladores.** 2009. Disponível em < http://pesquompile.wikidot.com/glassfish >. Acesso em 05 mai. 2011, 19:15.

HEFFELFINGER, David. **Java EE 6 with GlassFish 3 Application Server.** 1 ed. Birmingham: Packt Publishing, 2010.

JAAS REFERENCE GUIDE. **JAAS Reference Guide**. 2001. Disponível em < http://download.oracle.com/javase/1.4.2/docs/guide/security/jaas/JAASRefGuide.html >. Acesso em 20 abr. 2011, 14:12.

JavaEE: Conceitos Básicos. **Conceitos de Java.** Disponível em < http://javasemcafe.blogspot.com/2010/08/aula-04082010-4tads-conceitos-de-java.html >. Acesso em 04 mai. 2011, 22:47.

JAVA EE 5 TUTORIAL. **Working with Realms, User, Groups, and Roles.** 2010. Disponível em < http://download.oracle.com/javaee/5/tutorial/doc/bnbxj.html >. Acesso em 08 mai. 2011, 18:35.

LANDIM, Wikerson. **O que é captcha?**. 2009. Disponível em < http://www.tecmundo.com.br/2861-o-que-e-captcha-.htm >. Acesso em 12 mar. 2011, 19:15.

LINGHAM, Vinny. **Top 20 Reasons why Web Apps are Superior to Desktop Apps**. 2007. Disponível em < http://www.vinnylingham.com/top-20-reasons-why-web-apps-are-superior-to-desktop-apps.html >. Acesso em 15 mai. 2011, 18:45.

MUSSER, John et al. **All that JAAS.** 2009. Disponível em < http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-jaas.html > Acesso em 29 abr. 2011, 20:47.

Oracle. **Java Authentication + Authorization Services (JAAS) for E-Business Suite (OpenWorld 2008 Recap).** 2008. Disponível em < http://blogs.oracle.com/stevenChan/entry/java_authentication_authorization_services_jaas_for_ebsl >. Acesso em 28 abr. 2011, 19:23.

POSTGRESQL. **Sobre**. 2011. Disponível em < http://www.postgresql.org.br/sobre >. Acesso em 10 mar. 2011, 13:20.

PostgreSQL Architectural Concepts. **Overview of PostgreSQL Architecture and Internals**. 2005. Disponível em < http://elibrary.fultus.com/technical/topic/com.fultus.redhat.database/manuals/admin/overview.html >. Acesso em 19 abr. 2011, 16:32.

RIBEIRO, Leandro. **A importância do Backup na administração de sistemas.** 2009. Disponível em < http://imasters.com.br/artigo/11174/linux/a_importancia_do_backup_na_administracao_de_sistemas/ >. Acesso em 23 abr. 2011, 09:16.

SOUZA, S. E. Vítor. **Novidades do GlassFish 3.1**. 2011. Disponível em < http://www.devmedia.com.br/articles/viewcomp_forprint.asp?comp=21124 >. Acesso em 05 mai. 2011, 21:36.

TELLES. **Segurança no PostgreSQL – Parte I: "A Saga".** 2008. Disponível em < http://www.midstorm.org/~telles/2008/05/02/seguranca-no-postgresql-parte-i-a-saga/ >. Acesso em 21 abr. 2011, 19:54.

Using JavaMail API with Glassfish and Gmail. **Java EE Notes: Using JavaMail API with Glassfish and Gmail**. 2010. Disponível em < http://javaeenotes.blogspot.com/2010/04/using-javamail-api-with-glassfish-and.html >. Acesso em 19 mai. 2011, 18:15.

VIANA, Fábio. **Tutorial de JAAS**. 2006. Disponível em < http://www.guj.com.br/articles/184 >. Acesso em 09 mar. 2011, 15:41.

Webmaster Toolkits. **Listing of mime types**. 2011. Disponível em < http://www.webmaster-toolkit.com/mime-types.shtml >. Acesso em 15 mai. 2011, 21:10.