MANUAL DEL PROGRAMADOR 00-01

"MASKANA: UN GESTOR DE CONOCIMIENTO PARA RECUPERACIÓN Y BÚSQUEDA INTELIGENTE DE TRABAJOS DE GRADO DE LA UNIVERSIDAD DE NARIÑO"

GRUPO DE INVESTIGACIÓN APLICADA EN SISTEMAS –GRIAS

INGENIERÍA DE SISTEMAS

UNIVERSIDAD DE NARIÑO

Versión 2.0

San Juan de Pasto. FEBRERO 2016

TABLA DE CONTENIDO

2 INTRODUCCIÓN

Este documento es una guía para los programadores de la herramienta Maskana, en cuanto a su arquitectura y clases. El manual contiene los siguientes capítulos: Descripción general del sistema, Arquitectura y Aspectos de diseño e implementación.

2.1 OBJETIVO

El presente manual es una guía práctica para la compresión y descripción de la arquitectura de la herramienta y está dirigido a programadores, con o sin experiencia. Para los programadores sin experiencia es una ayuda práctica para adquirir conocimientos en la construcción de la herramienta, y para programadores con experiencia, este funcionara como documento de consulta.

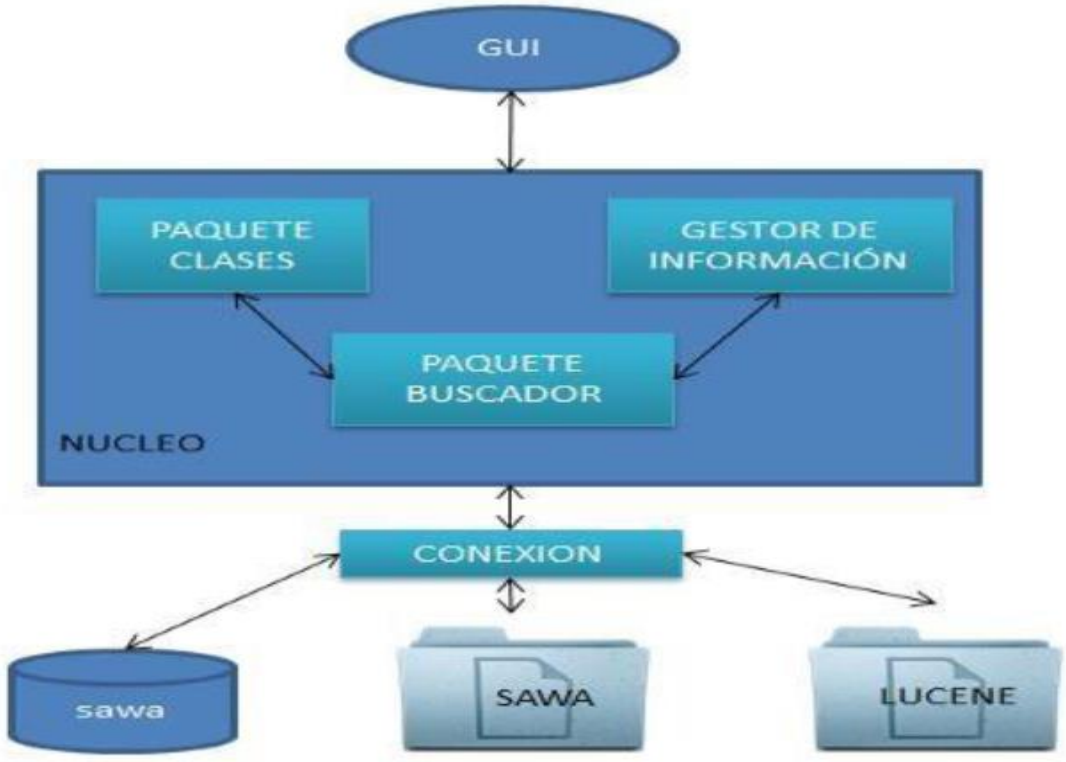
2.2 ALCANCE DEL DOCUMENTO

Este manual está dirigido a programadores del sistema Maskana - un gestor de conocimiento para recuperación y búsqueda inteligente de trabajos de grado en la Universidad de Nariño, se describe la arquitectura y aspectos de diseño e implementación de la herramienta.

3 ARQUITECTURA MASKANA

MASKANA es una herramienta de gestión de conocimiento soportada en una ontología dinámica, débilmente acoplada con un gestor de base de datos que le permite la búsqueda inteligente y recuperación eficiente de documentos digitales los cuales se encuentran almacenados y parametrizados en un repositorio general en la biblioteca “Alberto Quijano Guerrero” de la Universidad de Nariño .

La arquitectura de MASKANA es modular, compuesta por tres grandes módulos: el módulo de interfaz gráfica de usuario GUI, el módulo KERNEL y el módulo de conexión.



3.1 Módulo de interfaz gráfica de usuario o vista.

Este módulo es la herramienta que por medio de la visualización de imágenes y objetos, permite al usuario hacer uso de la herramienta de búsqueda para realizar consultas relacionadas con el dominio establecido por la ontología, además es importante ofrecer una interfaz gráfica amigable con el usuario, ofreciéndole facilidad en el momento de realizar la consulta.

Así este módulo da soporte gráfico a todos los demás módulos que lo requieran, estableciendo una conexión en doble vía entre el submódulo de Gestor de Información del módulo Núcleo y el GUI (del inglés *Graphic User Interface*).

3.2 Módulo núcleo.

A este módulo lo componen los paquetes de gran relevancia del proyecto, aquí se encuentran los algoritmos de búsqueda de información, de procesamiento de información, de presentación de resultados, procesos de gestión de información tanto del dominio establecido por la ontología así como también de usuarios que administraran la herramienta de búsqueda. Como su nombre lo indica este módulo es el núcleo de la herramienta de consulta y por tal motivo en él se albergan los paquetes que permiten el correcto funcionamiento de la herramienta. A continuación, se describen los submódulos que componen al módulo:

- **Paquete clases:** Este submódulo contiene las clases que representan la ontología y cada una de las tablas de la base de datos, que posteriormente servirán para gestionar el conocimiento con la ayuda de la herramienta, ofreciendo un orden claro y lógico de la representación de la ontología mediante las clases. Cabe mencionar que en este submódulo se representa la base del conocimiento de las clases del modelo y tiene una relación directa de doble dirección con el submódulo denominado Algoritmos de Búsqueda.

- **Gestor de información**: Este submódulo se encarga de gestionar información entre la Interfaz Gráfica de Usuario GUI con el submódulo de búsqueda, y de esta manera establecer la comunicación entre el módulo Núcleo y el módulo GUI.

- **Paquete buscador**: Este submódulo se encarga de albergar las clases que realizaran el proceso de la recuperación de información correspondiente a la

solicitud de búsqueda generada por el usuario y algoritmos de minería web encargados de obtener reglas de asociación descubriendo nuevo conocimiento en las descargas que realizan los usuarios. Este submódulo es de gran importancia ya que se encarga de realizar las consultas sobre la ontología haciendo uso del lenguaje SPARQL o sobre el Índice, dependiendo del tipo de consulta solicitada por el usuario. Este submódulo tiene comunicación directa y de doble sentido con los dos submódulos más que componen el módulo Núcleo, permitiendo así el intercambio de información.

3.3 Módulo de conexión

Como su nombre lo indica, este módulo es el encargado de realizar la respectiva conexión entre la base de datos, la ontología y el índice de búsqueda y de esa manera proveer al usuario información con características de persistencia. Con la implementación de este módulo se pretende dar respuestas eficientes y eficaces a las diferentes consultas ingresadas por el usuario. Se conecta con los siguientes elementos:

- ***Base de datos****:* En este submódulo la base de datos debe tener un diccionario, vocabulario o glosario de términos que represente la terminología de la ontología así como también se deben tener tablas relacionales al igual que los sistemas transaccionales para la gestión de usuarios, logs en sus consultas, roles y permisos.

- **Ontología**: Se almacena la ontología del dominio de documentos digitales donde se encuentra el conocimiento y las relaciones del dominio para su recuperación e inferencia.

- **Índice:** En este elemento se almacena todo el contenido textual indexado en un sistema de archivos para su eficiente recuperación teniendo en cuenta documentos en diferentes formatos (pdf,word,txt,etc) según sea el caso.

4 Aspectos de Diseño e Implementación de MASKANA

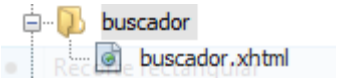
La herramienta MASKANA se diseñó utilizando la metodología ágil SCRUM , utilizando notación UML, siguiendo el patrón de arquitectura de software MVC (modelo de vista controlador) con el framework Java Server Faces, el cual facilita la mantenibilidad de la herramienta a futuros cambios. Su diseño es modular con el fin de permitir el crecimiento continuo de esta herramienta. MASKANA contiene algoritmos de búsqueda basados en el lenguaje SPARQL que se encargan de consultar la ontología Sawa y extraer el conocimiento acerca de una consulta. MASKANA se encuentra en un repositorio de Gibhub para el control de versiones.

MASKANA se desarrolló bajo el lenguaje de programación Java™ en su versión 1.7, lo que la convierte en una herramienta independiente a la plataforma donde se ejecute. Para su construcción, se usaron herramientas de software libre tales como el IDE de desarrollo Netbeans en su versión 7.2 , JSF con la biblioteca Primefaces en su versión 5 , API PDFBOX para la extracción de archivos con formatos .pdf, el sistema gestor de bases de datos PostgreSQL 9 con sus extensiones PGSIMILARITY para determinar similitud entre cadenas ,la librería JENA para la gestión de ontologías y finalmente se utilizó la librería LUCENE para administrar el índice de búsqueda.

4.1 Implementación módulo de interfaz gráfica de usuario o vista.

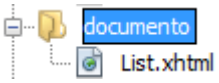
Específicamente este módulo está compuesto por todas las paginas xhtml de JSF (Java Server Faces) las cuales se encargan de interactuar con el usuario y llevar y traer información a los controladores que hacen parte del núcleo de Maskana. A continuación, se describen las páginas que forman el módulo de interfaz gráfica:

En la carpeta Buscador se encuentran vista:

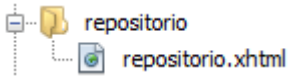


Buscador.xhtml: esta vista se encarga de permitir que el usuario realice búsquedas en el repositorio de documentos digitales almacenados en Maskana , tambien se encarga de presentar los resultados de una búsqueda realizada, permitiendo a su vez seguir realizando búsquedas a los documentos almacenados en Maskana y mirar en detalle un documento seleccionado por el usuario presentando la información de sus meta datos o descriptores.

En la carpeta documento se encuentra la vista:

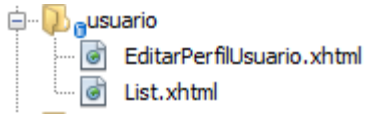


List.xhtml: esta vista permite al usuario gestionar los documentos digitales del repositorio de la biblioteca, así como también parametrizar los meta-datos y los tipos de documentos del gestor.



En la carpeta repositorio se encuentra la vista:

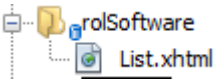
Repositorio.xhtml: esta vista permite al usuario administrador inicializar el repositorio digital, para que la herramienta permita recuperar información de los documentos.



En la carpeta usuario se encuentran las vistas:

EditarPerfilUsuario.xhtml: esta vista permite al usuario modificar su información personal en la herramienta, y definir una nueva contraseña.

List.xhtml: esta vista permite al usuario administrador asignar roles a otros usuarios y crear usuarios internos de acuerdo al perfil que requiera.



En la capeta rolSoftware se encuentra la vista:

List.xhtml: esta vista permite consultar, crear, eliminar y modificar los roles de los usuarios en la herramienta Maskana.

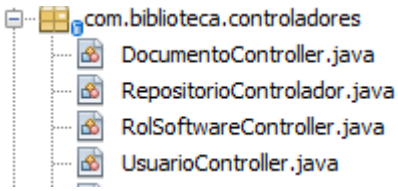
4.2 Implementación módulo núcleo.

En este módulo se reúnen los paquetes de gran relevancia del proyecto, aquí se encuentran los algoritmos de búsqueda de información, de procesamiento de información, de presentación de resultados, minería web, procesos de gestión de información tanto de documentos así como también de usuarios que administrarán la herramienta Maskana.

Como su nombre lo indica este módulo es el núcleo de la herramienta Maskana y por tal motivo en él se alberga el gestor de información, la base de datos mapeada dentro del submódulo Paquete Clases y el submódulo Paquete Buscador, que permiten el correcto funcionamiento de la herramienta. A continuación, se hace una descripción de los submódulos que componen el núcleo del sistema.

4.2.1 Gestor de información

Esta modulo lo compone el paquete controlador el cual tiene las siguientes clases:



DocuementoController.java: esta clase se encarga de crear, modificar, eliminar y consultar los tipos de documentos y asociarlos con los meta datos establecidos. Indexa los documentos en el repositorio digital de la biblioteca. Contiene los siguientes métodos

create():método para crear un documento en el repositorio.

update():método para editar un documento del repositorio.

destroy():método para eliminar un documento del repositorio.

iniciar():método para visualizar los documentos del repositorio.

RepositorioControlador.java: esta clase permite inicializar y conectarse al repositorio de la biblioteca Alberto Quijano Guerrero para la posterior indexación y recuperación de documentos. Contiene los siguientes métodos

crearRepositorio():método que permite iniciar y conectar el repositorio de búsqueda.

vaciarRepositorio():método que permite eliminar un repositorio.

RolSoftwareController.java: esta clase se encarga de gestionar la información acerca de los roles o perfiles de los usuarios en la herramienta.

UsuariosController.java: es la clase que permite manipular y autenticar la información de usuarios en la herramienta Maskana. Contiene los siguientes métodos

Créate():método encargado de crear un nuevo rol en el sistema.

Update():método encargado de actualizar un rol ya creado en el sistema.

Destroy():método encargado de eliminar un rol creado en el sistema.

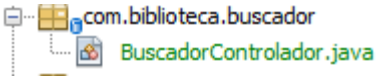
UsuarioController.java: Esta clase se encarga del manejo de usuarios internos como externos al sistema. Tiene los siguientes métodos:

Créate():método que permite crear un usuario en el sistema.

Update():método que permite editar la información de un usuario existente.

Destroy():método que permite eliminar un usuario del sistema.

4.2.2 Paquete Buscador



En este paquete se albergan algoritmos reutilizados y algoritmos desarrollados de acuerdo a las necesidades del sistema. Este paquete está compuesto por la clase BuscadorController que contiene los siguientes algoritmos:

Código del algoritmo JARO WINKLER: Encargado de encontrar palabras similares de una cadena de texto

```
query = query.replaceAll("'", "");
 List<String> resul = new ArrayList<String>();
 List<Object[]> listaWords;
 String[] listaTitulo = query.split(" ");
 String res = "";
 for (int i = 0; i < listaTitulo.length - 1; i++) {
     if (i == 0) {
         res = listaTitulo[i];
     } else {
         res = res + " " + listaTitulo[i];
     }
 }
 listaWords = this.vocabularioFacadel.findAllJaroWordsComplet
 (listaTitulo[listaTitulo.length - 1]);
 for (int i = 0; i < listaWords.size(); i++) {
     if (res.equals("")) {
         resul.add(listaWords.get(i)[0].toString());
     } else {
         resul.add(res + " " + listaWords.get(i)[0].toString());
     }

 }
 return resul;
```

Código del algoritmo Maskanita 2.0: encargado de consultar en la ontología SAWA actualizada a documentos digitales y retornar los de acuerdo a la cadena de búsqueda.

```
String[] busqueda = limpiarCadena(cadena busqueda);//metodo encargado de
limpiar la cadena de de busqueda;
String filtro = "";
this.palabra = "";
for (int i = 1; i < busqueda.length; i++) {
    this.palabra = this.palabra + " " + busqueda[i];
    if (i != busqueda.length - 1) {
        filtro = filtro + "REGEX(str(?sin)," + "\"" + busqueda[i] + "\",\"i\")|
    } else {
        filtro = filtro + "REGEX(str(?sin)," + "\"" + busqueda[i] + "\",\"i\")"
    }
}
String consulta = "PREFIX po1:<http://www.owl-ontologies.com/TesisGrado.owl#>
        + "select  "
        + "?id_tg?Titulo?Trabajo_grado (count(?id_tg)as ?c) "
        + " where{ "
        + "?Trabajo_grado po1:tiene ?keyword . "
        + "?Trabajo_grado po1:titulo?Titulo. "
        + "?Trabajo_grado po1:id_trabajo?id_tg. "
        + "?keyword po1:sinonimo ?sin. "
        + "FILTER ( "
        + filtro
        + ") "
        + "} "
        + "group by ?id_tg?Titulo?Signatura_Topografica?resumen?Trabajo_grado";
this.lista = new ArrayList<Tesis>();
this.lista=prepararLista(consulta);//metodo que ejcuta consulta sparql y
realiza el ranking de los trabajos de grado de acuerdo a la consulta
```

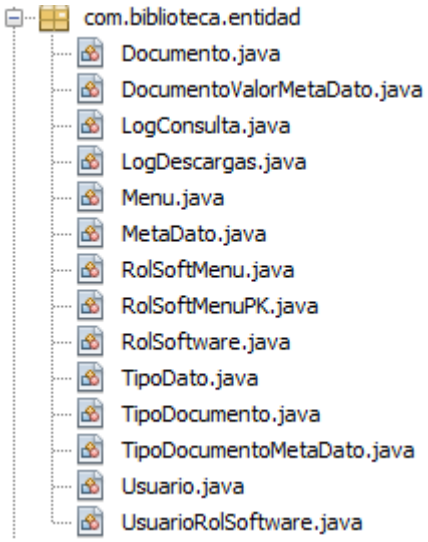
Codigo del algoritmo Apriori: Encargado de asociar documentos de acuerdo a las descargas realizadas por los usuarios**.**

```
public void regla(ItemSetFrequenteL l){
    String[] lItem=l.getL().replaceAll("]", "").replaceAll("\\[", "").split(",");
    for(int i=0;i<lItem.length;i++){
        String[] subL=new String[lItem.length-1];
        int c=0;
        for(int j=0;j<lItem.length;j++){
            if(!lItem[i].equals(lItem[j])){
                subL[c]=lItem[j].trim();
                c++;
            }
        }
        String subS= Arrays.toString(subL);
        double sopR=ap.getSoporte()/ap.getReglaLMap().get(subS.trim()).doubleValue();
        if(sopR>minConf){
            ReglaDto rF=new ReglaDto(subS,lItem[i] , sopR);
            rF.setSoporteRegla(ap.getReglaLMap().get(l.getL()).doubleValue());
            reglasFuertes.add(rF);
            antecendente.put(lItem[i].trim(), subL);
        }
    }
}
```

4.2.3 Paquete clases

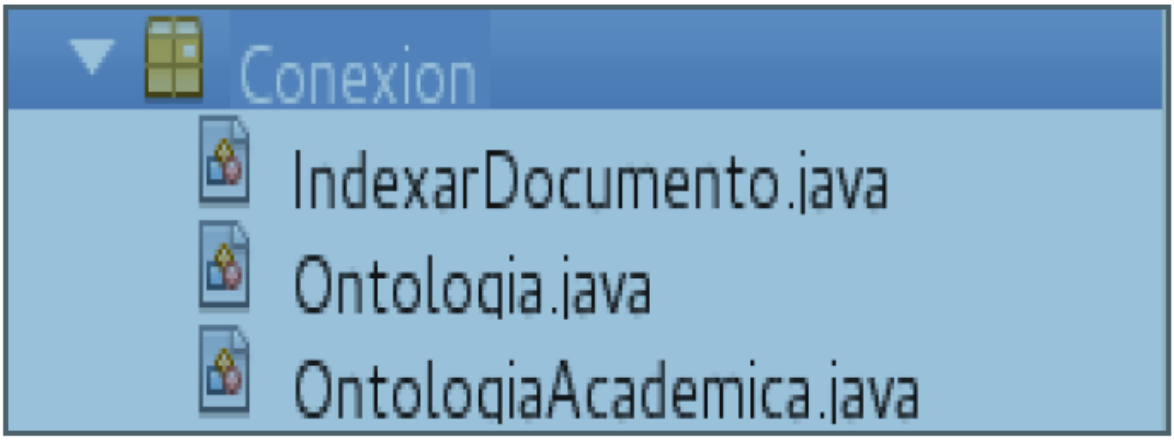
En este submódulo están todas y cada una de las clases que representan la ontología Sawa y cada una de las tablas de la base de datos que también lleva el mismo nombre, son utilizadas para la gestión del conocimiento con la ayuda de la herramienta, ofreciendo un orden claro y lógico de la representación de la ontología mediante las clases. Cabe mencionar que en este submódulo se representa la base del conocimiento de las clases del modelo y tiene una relación directa de doble dirección con el submódulo denominado Buscador.

El paquete de clases está compuesto por:



4.3 Implementación módulo de conexión.

Como su nombre lo indica, este módulo es el encargado de realizar la respectiva conexión entre la base de datos (Sawa), la ontología (SAWA) y el índice de búsqueda (LUCENE) y de esa manera proveer al usuario información con características de persistencia. Con la implementación de este módulo se pretende dar respuestas eficientes y eficaces a las diferentes consultas ingresadas por el usuario.



IndexarDocumento.java: esta clase se encarga de establecer comunicación con el índice y la ontología, permitiendo así la creación y actualización del índice de búsqueda y la ontología.
A continuación, se describen sus métodos:

indexarTexto(): método que permite indexar el contenido de un documento al índice de búsqueda.
modificarIndice(): método que permite actualizar el contenido de un documento al índice de búsqueda.
crearDocuementoOntologia():método que permite guardar el conocimiento del documento en la ontología.
modificarDocumento():método que permite actualizar el conocimiento del documento en la ontología.

Ontologia.java: esta clase se encargada de iniciar la conexión con la ontología equivalente al repositorio digital de la biblioteca Alberto Quijano Guerrero.

OntologiaAcademica.java: esta clase permite realizar consultas en SPARQL en la ontología.

5 Instalación de la aplicación

5.1 Instalación Postgres

Para poder instalar la aplicación se debe instalar PostgreSQL 9.1, Glassfish 3.1.2.2 y la extensión para PostgreSQL llamada pg_similarity, la aplicación solo puede ser instalada bajo sistemas operativos GNU / Linux

Para la instalación de PostgreSQL 9.1 (para Debian/ Linux), en una terminal se ejecuta lo siguiente:

```
$ su
# apt – get install postgresqlpostgresql – client pgadmin3
Postgresql –server –dev –all  postgresql –contrib
```

Ahora se borra la contraseña para la cuenta administrador de “postgres”, ejecutando la siguiente línea de comandos.

```
# su postgresql   –c   psql  template1
```

```
template1 =#ALTER USER postgresWHIT PASSWORD  ‘postgres1’ ;
template1 =#\q
```

Eso altera la contraseña dentro de la base de datos, ahora se tiene que hacer lo mismo para el usuario „Postgres“ y colocar la misma contraseña que utilizó anteriormente.

```
# passwd – d  postgres
 # su   postgres  – c  passwd
```

Ahora para instalar pg_similarity, primero se descarga desde github de la siguiente manera.

```
$ git   clone  https:// github.com/eulerto / pg_similarity.git
```

Luego para que pueda ser usada la extensión, se compila como superusuario de la siguiente manera.

```
# su
# cd  pg _ similarity
# USE_PGXS=1   make
# USE_PGXS=1   make   install
# supostgres
$ created bd_biblioteca
$ psql  –f  pg_similarity. Sql
```

Luego se restaura la copia de la base de datos bd_biblioteca.sql.

5.2 Instalación de Glassfish.

Primero se descarga la versión 3.1.2.2 para GNU/ linux desde la página de Oracle y se la ejecuta.

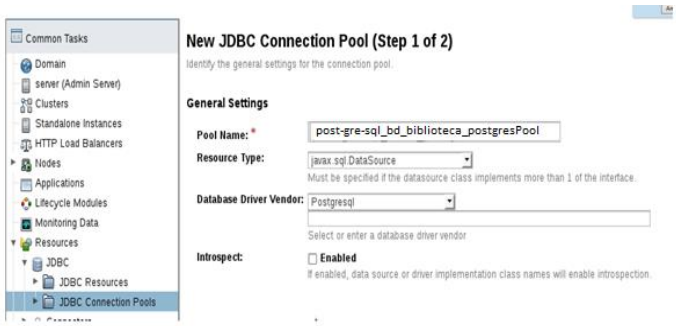
```
$  sh  ogs  – 3.1.2.2  –  unix  – ml.sh
```

Luego se descarga el driver JDBC de PostgreSQL desde la página de PostgreSQL y copiarlo en el directorio de glassfish3/glassfish/domains/domain1/lib. Para iniciar el servidor se ejecuta siguiente.

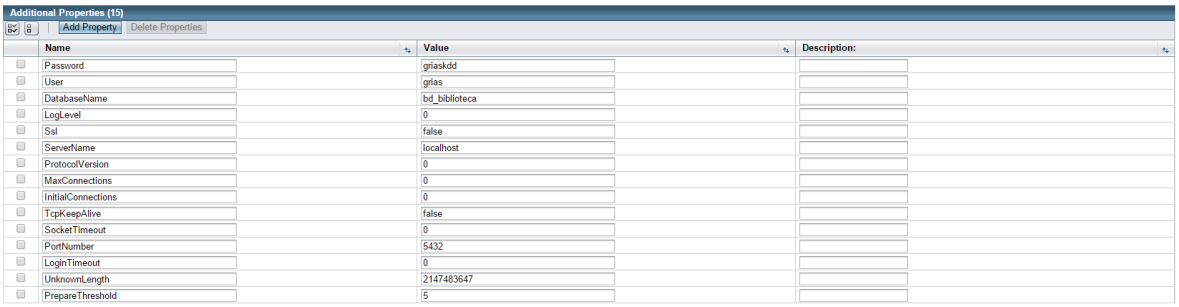
```
$ ./glassfish3/glassfish/bin/startserv
```

Con esto ya en el navegador se ingresa con la dirección http://localhost:8080, y se entra al a consola de administración, se ingresa usuario y contraseña de haber escrito una en la instalación de glassfish.

Ir a “Resources/JDBC/Conection Pools” y crear una nueva conexión con los datos que muestra la figura y luego clic en siguiente.



Seleccione el origen de los datos de nombre de clase org.postgresql.ds.PgconnectionPoolDataSource y escribir a las siguientes propiedades adicionales como muestra siguiente figura.



Ya con esto se guarda las conexiones y damos clic en finalizar.

Luego en “Resources/JDBC Resources” se escribe el nombre JNDI y se escoge el Pool Name creado anteriormente como lo muestra la siguiente figura.

Finalmente se procede a publicar la aplicación