JGenea Web 2.1 - Guide Utilisateur

L'application généalogique 100% java

Templth

JGenea Web 2.1 - Guide Utilisateur: L'application généalogique 100% java

Templth

Publié $Date: 2004/02/17 14:00:16 $

Ce guide d'utilisation est destiné aux personnes désirant utiliser l'application web de JGenea version 2.1 pour leurs travaux généalogiques (consultation de leur généalogie, consultation de la gestion de leurs recherches, visualisation de registres numérisés et tout çà en ligne). Il détaille dans un premier temps la mise en place des prérequis nécessaires pour faire fonctionner l'application. Nous détaillerons l'installation sous Linux mais celle-ci pourra être généraliser pour n'importe quel système d'exploitation. Dans un deuxième temps, il sera détailler l'installation et la configuration de l'application (bases de données utilisées, répertoires...).

La version 2.1 de JGenea (version web) se base sur la même couche d'accès aux données que l'autre version. Elle a été optimisée pour gagner en performance sur l'affichage des registres numérisés.

Contrairement à la version IHM de JGenea, il est nécessaire d'avoir quelques notions d'informatiques pour installer l'application et connaître Linux.

Si vous remarquez des erreurs ou des points non suffisament détaillés, n'hésitez pas à me le faire savoir pour améliorer ce guide d'utilisation. Celui-ci sert de point de départ; par la suite, il est recommandé d'utiliser les forums de discussions pour d'éventuelles questions, suggestions.

Table des matières

Liste des illustrations

I. Prérequis

JGenea Web est une application web de généalogie écrite en java. Elle est open source: cela signifie qu'il est libre d'utilisation.

Comme toute application web java, elle ne peut fonctionner toute seule et doit être déployée dans un serveur d'application Java qui se charge de transmettre les requêtes HTTP à l'applicatif entre autres.

De plus JGenea Web ne fournit pas de base de données contrairement à JGenea Ihm, mais, comme le modèle est disponible dans le cvs, il est possible de créer une base sur laquelle se baser.

Ce document va détailler la mise en place de JGenea Web sur une machine ayant pour système d'exploitation Linux. Nous utiliserons la distribution Debian pour ces facilités d'administration et de gestion des packages. Il est donc indispensable d'avoir cette distribution correctement installée avant de commencer...

Cette partie va donc détailler l'installation des différents composants indispensables au bon fonctionnement de JGenea Web:

- Une machine virtuelle java (jre ou jdk),
- Un display,
- Une base de données,
- Un serveur web,
- Un serveur applicatif java (moteur de servlets),
- Un module faisant le lien entre le serveur web et le serveur applicatif java,
- La bibliothèque native pour les graphiques.

Figure 1. Architecture de JGenea Web

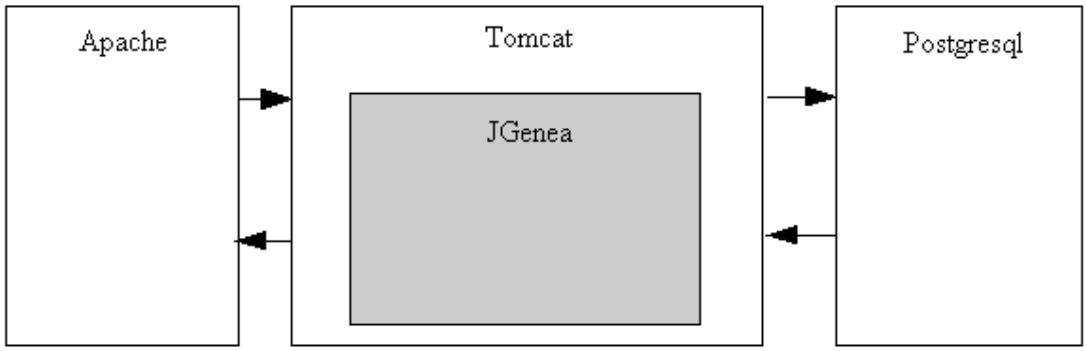

Table des matières

1. Sources des packages

Pour pouvoir utiliser apt-get, l'outil de gestion des packages, il faut configurer correctement les sources pour trouver les packages que l'on désire utiliser.

Voici un exemple de fichier `/etc/apt/sources.list`:

```
# See sources.list(5) for more information, especialy
# Remember that you can only use http, ftp or file URIs
deb http://http.us.debian.org/debian stable main contrib non-free
deb http://non-us.debian.org/debian-non-US stable/non-US main contrib non-free
deb http://security.debian.org stable/updates main contrib non-free

# jdk1.3
deb ftp://metalab.unc.edu/pub/linux/devel/lang/java/blackdown.org/debian woody non-free
```

Attention

Les sites sources peuvent parfois changer. Pour retrouver les nouveaux, il suffit de faire des recherches avec les noms de packages situés par la suite...

2. Configuration des langues

Sous Linux, dpkg-reconfigure locales

3. JDK / JRE

JGenea doit obligatoirement utiliser une machine virtuelle Java pour pouvoir fonctionner. Il s'agit d'un programme permettant de lancer les applications écrites en Java.

Celle-ci se trouve sous deux formes: jdk (Java Development Kit) ou jre (Java Runtime Environment). Il est conseillé d'installer la deuxième car elle prend moins de place à moins que vous n'ayez envie de faire des développements en Java.

Nous allons utiliser les jdk / jre fournis par blackdown. Le(s) package(s) à installer sont j2sdk1.3 (j2re1.3). La commande pour installer est la suivante:

```
kerion:~# apt-get install j2sdk1.3
```

4. Display

Java, par défaut, utilise les bibliothèques natives pour afficher des éléments graphiques. Cela sous-entend qu'il faut un display, c'est-à-dire un programme qui affiche les éléments graphiques. Cependant sur la plupart des serveurs, il y a pas de display...

Une des solutions est d'utiliser d'autres bibliothèques que celles natives. PJA est une implémentation graphique gratuite disponible, mais elle est assez difficile à mettre en place car il faut spécifier des paramètres à la machine virtuelle.

Une autre solution est d'utiliser un display "virtuel", c'est-à-dire un programme qui en simule un. X Virtual Frame Buffer (un programme de XFree 86) permet de faire cela. Sous Debian, il est très simple à utiliser. Pour qu'il démarre, il faut par contre installer en plus les polices.

Voici les packages des polices nécessaires:

```
kerion:~# dpkg -l | grep xfont
ii  xfonts-100dpi  4.1.0-16woody1 100 dpi fonts for X
ii  xfonts-75dpi   4.1.0-16woody1 75 dpi fonts for X
ii  xfonts-abi     1.0.2+cvs.2002 Some fonts shipped with AbiWord
ii  xfonts-base    4.1.0-16woody1 standard fonts for X
ii  xfonts-pex     4.1.0-16woody1 fonts for minimal PEX support in X
ii  xfonts-scalabl 4.1.0-16woody1 scalable fonts for X
```

La commande pour installer est la suivante:

```
kerion:~# apt-get install xvfb
```

5. Base de données

Comme base de données, nous n'allons pas utiliser la base de données de JGenea IHM qui est une base de données java, Hypersonic. Il est préférable d'utiliser Postgresql qui est plus performante, a des fonctionnalités supplémentaires à Hypersonic et des outils en ligne de commande.

Pour installer Postgresql, il suffit d'utiliser la commande suivante:

```
kerion:~# apt-get install postgresql
```

Cela installe également les outils clients de la base de données.

```
kerion:~# dpkg -l | grep postgresql
ii  postgresql     7.2.1-2woody2  Object-relational SQL database, descended fr
ii  postgresql-cli 7.2.1-2woody2  Front-end programs for PostgreSQL
```

Pour pouvoir utiliser Postgresql avec le driver JDBC, il faut que le moteur de base de données soit lancé en mode réseau car celui est de type réseau. Cela se réaliser en ajoutant la paramètre -i sur le lancement de **postmaster**.

Sinon le driver jdbc (`jdbc7.1-1.2.jar`) est disponible dans la distribution de JGenea Web. Il suffit de copier celui dans le répertoire `common/lib` de la distribution de Tomcat pour que celui-ci soit pris en compte.

Attention

Il est impératif d'utiliser le driver correspondant à la version de Postgresql utilisée sous peine que la connexion ne s'établisse pas.

6. Serveur Web

Pour des raisons de performance, il est préférable placer un serveur web avant le serveur applicatif java. Apache nous paraît être un bon serveur web gratuit et très répandu. Il existe une version HTTP et une version gérant SSL pour des connexions sécurisées. En fait, sans SSL, il faut être conscient que les identifiants de connexion et mots de passe passent en clair sur le réseau!

Il existe deux packages différents suivants le serveur que l'on veut utiliser (SSL ou non): apache et apache-ssl. Voici les commandes pour les installer:

```
kerion:~# apt-get install apache
```

et

```
kerion:~# apt-get install apache-ssl
```

Note

Lors de l'installation d'apache-ssl, vous devez renseigner les informations concernant le certificat que vous allez utiliser.

7. Serveur applicatif Java

Nous allons utiliser comme serveur d'application java, Tomcat en version 4.1.29. Tomcat est un serveur robuste open-source qui peut être utilsé seul (avec son serveur HTTP interne, mais c'est peu récommandé) ou couplé avec un serveur HTTP qui lui fait suivre les requêtes qu'il reçoit (c'est recommandé car plus performant). Pour faire suivre les requêtes, nous allons utilisé un module apache.

7.1. Installation

Dans le cas de Tomcat, nous n'allons pas utiliser l'outil de Debian, mais nous allons télécharger la distribution directement sur le site mirroir d'Apache: http://mir2.ovh.net/ftp.apache.org/dist/jakarta/tomcat-4/v4.1.29/. L'installation consiste uniquement en la compression du fichier (ayant pour extension tar.gz pour Linux).

La distribution se compose de plusieurs répertoires dont ceux-ci:

- `bin/` : le répertoire contenant les scripts de lancement, arrêt... de Tomcat,
- common/ : le répertoire contenant les bibliothèques java utilisées par Tomcat. Il faudra copier les bibliothèques des drivers jdbc utilisés dans le sous-répertoire `lib/`,
- conf/ : le répertoire contenant la configuration de Tomcat et notamment le fichier `server.xml`, que nous détaillerons par la suite.
- logs/ : le répertoire où se trouve toutes les logs de Tomcat (le fichier catalina.out est la sortie standard de Tomcat),
- webapps/ : le répertoire dans lequel il faudra copier l'application web de JGenea Web (fichier jgenea.war)/

7.2. Configuration

Pour pouvoir lancer Tomcat, il faut éditer le fichier <TOMCAT_HOME>/bin/catalina.sh et positionner les variables de la manière suivante:

```
JAVA_HOME=/usr/lib/j2sdk1.3
JAVA_OPTS=-Xmx96m
```

Note

JAVA_HOME est le chemin d'installation du jdk / jre. $JAVA_HOME/bin/java doit être la machine virtuelle que vous allez utiliser.

Note

Dans la variable JAVA_OPTS, est précisé un paramètre pour augmenter la taille mémoire de la machine virtuelle. Cela évite les plantages de type OutOfMemory lors de l'utilisation de JAI avec des images de grandes tailles.

Dans un second temps, il faut configurer le serveur pour qu'il se mette en attente des requêtes que va lui envoyer Apache (le serveur Web).

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">

  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener"
            debug="0"/>
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"
            debug="0"/>

  <!-- Define the Tomcat Stand-Alone Service -->
  <Service name="Tomcat-Standalone">

    <!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
    <Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
               port="8009" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="0"
               useURIValidationHack="false"
               protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>

    <!-- Define the top level container in our container hierarchy -->
    <Engine name="Standalone" defaultHost="localhost" debug="0">
      <!-- Global logger unless overridden at lower levels -->
      <Logger className="org.apache.catalina.logger.FileLogger"
              prefix="catalina_log." suffix=".txt"
              timestamp="true"/>

      <!-- Define the default virtual host -->
      <Host name="localhost" debug="0" appBase="webapps"
       unpackWARs="true" autoDeploy="true">

        <Logger className="org.apache.catalina.logger.FileLogger"
                 directory="logs"  prefix="localhost_log." suffix=".txt"
                 timestamp="true"/>
      </Host>

    </Engine>

  </Service>

</Server>
```

Ce fichier définit que la port 8009 est le port utilisé par le connecteur JK2 que nous utilisons pour les requêtes ayant été envoyés par Tomcat. Ce port devra être configuré dans Apache pour qu'il sache vers où faire suivre les requêtes.

Note

Il est possible que Tomcat et Apache ne soit pas sur les mêmes machines.

Il est également possible de définir des pools de connexion pour les bases de données dans ce fichier. Cela se fait de la manière suivante:

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">

  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener"
            debug="0"/>
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"
            debug="0"/>

  <!-- Define the Tomcat Stand-Alone Service -->
  <Service name="Tomcat-Standalone">
```

```
    <!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
    <Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
               port="8009" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="0"
               useURIValidationHack="false"
               protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>

    <!-- Define the top level container in our container hierarchy -->
    <Engine name="Standalone" defaultHost="localhost" debug="0">
      <!-- Global logger unless overridden at lower levels -->
      <Logger className="org.apache.catalina.logger.FileLogger"
              prefix="catalina_log." suffix=".txt"
              timestamp="true"/>

      <!-- Define the default virtual host -->
      <Host name="localhost" debug="0" appBase="webapps"
       unpackWARs="true" autoDeploy="true">

        <DefaultContext debug="99">
          <Resource name="jdbc/JGenea" auth="Container"
                    type="javax.sql.DataSource"/>
          <ResourceParams name="jdbc/JGenea">
            <parameter>
              <name>factory</name>
              <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
            </parameter>
            <parameter><name>username</name><value>sa</value></parameter>
            <parameter><name>password</name><value></value></parameter>
            <parameter><name>driverClassName</name>
              <value>org.hsqldb.jdbcDriver</value></parameter>
            <parameter><name>url</name>
              <value>jdbc:hsqldb:hsql://localhost:66</value></parameter>
            <parameter>
              <name>maxActive</name>
              <value>20</value>
            </parameter>
            <parameter>
              <name>maxIdle</name>
              <value>10</value>
            </parameter>
            <parameter>
              <name>maxWait</name>
              <value>-1</value>
            </parameter>
          </ResourceParams>
        </DefaultContext>

      </Host>

    </Engine>

  </Service>

</Server>
```

Note

Pour pouvoir utiliser JGenea Web, il n'est pas forcément nécessaire de créer un pool de connexion au niveau du serveur d'application comme nous le verrons par la suite.

8. Lien entre les serveurs Web et d'application Java

Pour faire le lien entre le serveur web (apache) et le serveur applicative java (tomcat), il est nécessaire d'utiliser un module apache: JK2. Il s'agit d'une bibliothèque C qui se trouve dans le fichier `mod_jk.so`.

Il est nécessaire de configurer apache pour qu'il prenne en compte le module en spécifiant quelques propriétés (application web impactée, format des logs, fichier de logs et fichiers de configuration du "worker").

Tout cela est à définir dans le fichier httpd.conf d'apache où se trouve toute la configuration du serveur. Il suffit de rajouter les lignes suivantes à la ligne de la partie définissant les modules à charger (LoadModule...):

```
LoadModule    jk_module  /usr/lib/apache/1.3/mod_jk.so
AddModule     mod_jk.c
JkWorkersFile /applis/jakarta-tomcat-4.1.18/conf/workers2.properties
JkLogFile     /applis/jakarta-tomcat-4.1.18/logs/mod_jk.log
JkLogLevel    info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
#JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
#JkRequestLogFormat "%w %V %T"
JkMount /jgenea/* worker1
```

Note

Il est à noter que les chemins peuvent être changés mais doivent correspondre aux chemins où se trouvent les différents éléments.

Attention

La dernière ligne ne peut être modifié car il ne s'agit pas d'un chemin.

Le fichier worker2.properties spécifié, permet de définir plus finement la communication entre les deux serveurs (ports...) pour le "worker" spécifié précédemment. Voici le code de ce fichier dans notre configuration:

```
# Define the communication channel
worker.list=worker1

# Set properties for worker1 (ajp13)
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=50
worker.worker1.cachesize=10
worker.worker1.cache_timeout=600
worker.worker1.socket_keepalive=1
worker.worker1.socket_timeout=300
```

9. Bibliothèque native pour les graphiques

JAI (Java Advance Imaging) est une bibliothèque java de manipulation des images qui est utilisée par JGenea. Il est possible d'utiliser la bibliothèque mediaLib native en C via un wrapper (interface JNI). Il est donc nécessaire que Java retrouve cette dernière bibliothèque native. Ceci se fait grâce à la variable d'environnement LD_LIBRARY_PATH sous Linux.

Nous allons donc la spécifier au début du script de lancement de Tomcat. Il suffit de mettre le chemin où se trouve le fichier `libmlib_jai.so`.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/applis/jai-1_1_2/lib/
```

II. Mise en oeuvre

Table des matières

10. Introduction

La distribution de JGenea Web contient le fichier jgenea.war qui est prêt à être déployé dans le serveur d'application java. Cependant celle-ci a été configuré avec des paramètres par défaut. Il est souvent nécessaire de changer ces paramètres. Pour ce faire, il faut extraire tous les fichiers du fichier war (Web ARchive) pour en modifier certains. Ceci se fait de la manière suivante avec l'utilitaire jar (inclu dans le jdk) :

```
kerion:~# jar xvf jgenea.war
```

Note

Cette commande extrait tous les fichiers dans le répertoire courant.

Les fichiers qui sont succeptibles d'être modifiés sont les suivants:

- WEB-INF/configuration.xml : configuration général de JGenea Web,
- WEB-INF/struts-config.xml : configuration de struts (framework applicatif sur lequel se base l'application web).

Pour recréer le fichier jgenea.war, il suffit de se placer dans le répertoire où on a fait l'extraction et d'exécuter la commande suivante:

```
kerion:~# jar cvf jgenea.war
```

Pour installer l'application, il suffit de copier ce fichier dans le répertoire `<TOMCAT_HOME>/webapps`. Lors du démarrage de Tomcat, celui-ci va être déployé.

Attention

S'il existe un répertoire `jgenea` dans `<TOMCAT_HOME>/webapps`, le fichier war ne sera pas redéployé... Pour qu'il soit redéployé, il suffit de supprimer l'ancien répertoire.

11. Base de données

Si l'on choisit d'utiliser Postgresql, il y a plusieurs à faire pour que cela fonctionne.

11.1. Accès à Postgresql

Tout d'abord, les drivers jdbc de Postgresql sont d'un type qui utilise forcément un accès réseau à la base même si l'application utilisant celui-ci se trouve sur la même machine que la base (via localhost ou 127.0.0.1). Il faut donc activer le mode réseau de Postgresql. Il s'agit en fait d'un paramètre de postmaster. Il s'agit plus particulièrement de l'option -i.

Pour ce faire, il suffit d'éditier le fichier `/etc/postgresql/postmaster.conf` et de mettre dans la variable POSTMASTER_OPTIONS, "-i".

```
# /etc/postgresql/postmaster.conf
#
# Copyright (c) Oliver Elphick 1997, 2001
# Part of the Debian package, postgresql. The Debian packaging is
# licensed under GPL v.2
#
# This is the configurable initialisation of the postgresql package
# The defaults are shown, but are commented out.
#
# As of release 7.1, many parameters may now be found in
# /etc/postgresql/postgresql.conf.  To avoid confusion, these can
# no longer be set here, even though the command line options that
# used to control them do still exist.
#

POSTGRES_HOME=`getent passwd postgres | awk -F: '{print $6}' | head -1`
if [ -z "$POSTGRES_HOME" ]
then
    POSTGRES_HOME=/var/lib/postgres
fi

# Where to find the PostgreSQL database files, including those that
# define PostgresSQL users and permissions.
# POSTGRES_DATA=/var/lib/postgres/data

# Any special options to pass to the postmaster through pg_ctl's -o option.
# This may include such options as "-h hostname", for which there is no
# parameter defined.  However most options can be set by editing
# postgresql.conf appropriately.
POSTMASTER_OPTIONS="-i"

# Minimum number of entries in the kernel file table.  If the table size is
# lower, postgresql.startup attempts to increase it by writing this parameter
# into /proc/sys/kernel/file-max.  This is only effective if the kernel has
# been compiled to support run-time configuration.
# KERNEL_FILE_MAX=1032


# Where to send logging and debugging traces.  By default, very little
# should appear here, because SYSLOG is set to 2 in postgresql.conf, so
# that all messages are sent to syslog only.
#
# If you change this, remember to change /etc/logrotate.d/postgresql too.
# POSTGRES_LOG=/var/log/postgresql/postgres.log
```

Dans un deuxième temps, il faut définir la manière dont on va s'authentifier à Postgresql en fonction de la manière dont on y accède. Ceci se configure dans le fichier `/etc/postgresql/pg_hba.conf`. On va considérer que si on essaie via le réseau (dans notre cas, en jdbc), Postgresql fera un authentification avec mot de passe et si on y accède en local, l'authentification se fera par rapport à l'utilisateur unix que l'on utilise.

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the PostgreSQL Administrator's Guide, chapter "Client
# Authentication" for a complete description.  A short synopsis
# follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of three forms:
#
# local    DATABASE  USER  METHOD  [OPTION]
# host     DATABASE  USER  IP-ADDRESS  IP-MASK  METHOD  [OPTION]
# hostssl  DATABASE  USER  IP-ADDRESS  IP-MASK  METHOD  [OPTION]
#
# (The uppercase quantities should be replaced by actual values.)
# DATABASE can be "all", "sameuser", "samegroup", a database name (or
# a comma-separated list thereof), or a file name prefixed with "@".
# USER can be "all", an actual user name or a group name prefixed with
# "+" or a list containing either.  IP-ADDRESS and IP-MASK specify the
# set of hosts the record matches.  METHOD can be "trust", "reject",
# "md5", "crypt", "password", "krb5", "ident", or "pam".  Note
# that "password" uses clear-text passwords; "md5" is preferred for
# encrypted passwords.  OPTION is the ident map or the name of the PAM
# service.
#
# This file is read on server startup and when the postmaster receives
# a SIGHUP signal.  If you edit the file on a running system, you have
# to SIGHUP the postmaster for the changes to take effect, or use
# "pg_ctl reload".

# Put your actual configuration here
# ----------------------------------
#
# This default configuration allows any local user to connect as himself
# without a password, either through a Unix socket or through TCP/IP; users
# on other machines are denied access.
#
# If you want to allow non-local connections, you need to add more
# "host" records before the final line that rejects all TCP/IP connections.
# Also, remember TCP/IP connections are only enabled if you enable
# "tcpip_socket" in /etc/postgresql/postgresql.conf.

# TYPE  DATABASE    USER        IP-ADDRESS        IP-MASK           METHOD

# DO NOT DISABLE!
# If you change this next entry you will need to make sure the postgres user
# can access the database using some other method.  The postgres user needs
# non-interactive access to all databases during automatic maintenance
# (see the vacuum command and the /usr/lib/postgresql/bin/do.maintenance
# script).
local   all         postgres                                        ident sameuser
host    all         postgres    127.0.0.1         255.255.255.255   password
host    all         postgres    localhost         255.255.255.255   password

local   all         all                                             ident sameuser
host    all         all         127.0.0.1         255.255.255.255   password
host    all         all         0.0.0.0           0.0.0.0           reject
```

11.2. Création de la base

JGenea Web ne permet pas de créer de base de données à partir de rien. Il faut donc, dans le cas de Postgresql, créer la base avec l'utilitaire createdb, puis créer le schéma et l'initialiser avec JGenea Ihm depuis la console.

Voici la commande pour créer la base:

```
kerion:~# createdb -E UNICODE jgenea-perso
```

Note

jgenea-perso est le nom de la base dans Postgresql.

Attention

Cette commande doit être lancée avec le même utilisateur que celui qui a lancé le serveur de base de données. Par défaut, il s'agit de l'utilisateur postgres.

Si on utilise la console de JGenea Ihm pour initialiser le schéma et la base, il faut le configurer correctement et ajouter le driver jdbc dans le classpath de celui-ci.

Voici le nouveau script de lancement de la console (`console.sh`):

```
#!/bin/sh
#export JAVA_HOME=/applis/jdk1.3.1_01

export CLASSP=../lib/jgenea-ihm.jar
export CLASSP=$CLASSP:../lib/jgenea-dao.jar
export CLASSP=$CLASSP:../lib/jgenea-etats.jar
export CLASSP=$CLASSP:../lib/jgenea-fusion.jar
export CLASSP=$CLASSP:../lib/jgenea-gedcom.jar
export CLASSP=$CLASSP:../lib/iText.jar
export CLASSP=$CLASSP:../lib/hsqldb.jar
export CLASSP=$CLASSP:../lib/jdbc7.1-1.2.jar
export CLASSP=$CLASSP:../lib/jakarta-regexp-1.2.jar
export CLASSP=$CLASSP:../lib/xerces.jar
export CLASSP=$CLASSP:../lib/xalan.jar
export CLASSP=$CLASSP:../lib/jaxp.jar
export CLASSP=$CLASSP:../lib/fop.jar
export CLASSP=$CLASSP:../lib/avalon-framework-cvs-20020806.jar
export CLASSP=$CLASSP:../lib/batik.jar
export CLASSP=$CLASSP:../lib/jai_codec.jar
export CLASSP=$CLASSP:../lib/jai_core.jar
export CLASSP=$CLASSP:../lib/log4j-1.2.7.jar
export CLASSP=$CLASSP:../lib/skinlf.jar
export CLASSP=$CLASSP:../lib/kunststoff.jar
export CLASSP=$CLASSP:../lib/ext/

java -Xmx128M -DJGENEA_CONFIGURATION_XML=
            /applis/jgenea/exploitation/ihm/configuration.xml
     -classpath $CLASSP org.jgenea.console.Console $1 $2 $3 $4
```

Note

Le fichier jar du driver jdbc est ici `jdbc7.1-1.2.jar`.

Le bloc de définition de la base dans le fichier de configuration configuration.xml relatif à la base précédemment citée est le suivant:

```
<connexion id="perso" defaut="non" type="db">
  <param nom="base">Postgresql</param>
  <param nom="descriptif">Base perso</param>
  <param nom="driver">org.postgresql.Driver</param>
  <param nom="url">jdbc:postgresql://localhost:5432/jgenea-perso</param>
  <param nom="login">postgres</param>
  <param nom="password">votremotdepasse</param>
  <param nom="factory-dao">org.jgenea.isolation.defaut.FactoryDAOImpl</param>
  <param nom="factory-connexion-dao">
    org.jgenea.isolation.defaut.ConnexionFactoryUniqueDAOImpl</param>
</connexion>
```

Note

Pour plus d'informations sur le fichier `configuration.xml`, se reporter au manuel utilisateur de JGenea Ihm.

11.3. Initialisation de la base

Après avoir créé la base, il faut créer le schéma de la base et remplir certains tables avec des informations. Ceci peut être fait de deux manières.

La première consiste à utiliser la console de JGenea Ihm pour exécuter cette tache.

```
kerion:~# ./console.sh
install -repertoire JGENEA_REPERTOIRE_SCRIPTS -type psql -pays fr,ita
```

La seconde consiste à exécuter les scripts de création de schéma et de remplissage directement depuis un client de Postgresql (psql).

```
kerion:~# psql jgenea-perso
\i crebas-psql.sql
\i departements-fr-psql.sql
\i communes-fr-psql.sql
\i departements-ita-psql.sql
\i communes-ita-psql.sql
\i liaison_mariage_pers_type-psql.sql
\i liaison_personnes_type-psql.sql
\i mariage_evt_type-psql.sql
\i personne_caract_type-psql.sql
\i personne_evt_type-psql.sql
\i type_acte-psql.sql
\q
```

11.4. Driver jdbc

Pour que le driver jdbc soit pris en compte par Tomcat, il suffit de placer le fichier jar (ici le fichier `jdbc7.1-1.2.jar`) dans le répertoire `<TOMCAT_HOME>common/lib/` avant le démarrage du serveur.

12. Configuration

La configuration se fait au maximum dans deux fichiers. Ils sont les suivants (ils ont déjà été cités précédemment) :

- WEB-INF/configuration.xml
- WEB-INF/struts-config.xml

12.1. Base de données

L'accès aux bases de données peut se faire de deux manières:

- Soit en utilisant les pools de connexion de Struts. Dans ce cas, la configu
- Soit en utilisant les pools de connexion du serveur d'application java.

12.1.1. Pools de Struts

Dans ce cas, les pools sont à configurer dans le fichier struts-config.xml. Seul le contenu sous la balise <data-source> nous intéresse. Celle-ci se trouve directement sous la balise <struts-config>.

Elle permet de lister les différents pools (data-source) utilisables par l'application.

Attention

Les différentes bases configurées doivent être disponibles au moment où le serveur d'application java démarre. Dans le cas contraire, l'application ne serait pas correctement démarrer et reverrait une erreur lors des tentatives d'accès.

```
<data-sources>
  <data-source key="perso">
    <set-property property="autoCommit" value="false" />
    <set-property property="description" value="Psql" />
    <set-property property="driverClass" value="org.postgresql.Driver" />
    <set-property property="maxCount" value="4" />
    <set-property property="minCount" value="2" />
    <set-property property="password" value="votremotdepasse"/>
    <set-property property="url"
                  value="jdbc:postgresql://localhost:5432/jgenea-perso" />
    <set-property property="user" value="postgres" />
  </data-source>
</data-sources>
```

12.1.2. Pools de connexion du serveur d'application java

Aucune configuration n'est nécessaire au niveau de JGenea pour créer ce type de pool puisqu'il est défini dans le serveur d'application. Pour plus d'information, se reporter au paragraphe la configuration du serveur d'application.

12.1.3. Accès aux différents pools

Une fois les pools configurés, il faut configurer l'application pour qu'elle puis y accéder. Cela se fait grâce au fichier `configuration.xml`.

Dans le cas d'un pool struts, l'élément de configuration est le suivant:

```
<!-- Liste des connexions -->
<connexions>
  <connexion id="perso" defaut="non" type="db">
    <param nom="base">Postgresql</param>
    <param nom="descriptif">Base perso</param>
    <param nom="nom-struts">perso</param>
    <param nom="factory-dao">
        org.jgenea.isolation.defaut.FactoryDAOImpl</param>
    <param nom="factory-connexion-dao">
        org.jgenea.isolation.j2ee.ConnexionFactoryPoolStrutsDAOImpl</param>
    <param nom="factory-images-dao">
        org.jgenea.images.registres.FactoryImagesDAOImpl</param>
    <param nom="cache-active">oui</param>
    <param nom="reload-image-pour-affichage">non</param>
    <param nom="repertoire-cache">.</param>
    <param nom="taille-maximum-repertoire-cache">2000000</param>
  </connexion>
</connexions>
```

Dans le cas d'un pool du serveur d'application java, l'élément de configuration est le suivant:

```
<!-- Liste des connexions -->
<connexions>
  <connexion id="perso2" defaut="non" type="db">
    <param nom="base">Postgresql</param>
    <param nom="descriptif">Base perso</param>
    <param nom="jndi-initial-context"></param>
    <param nom="jndi-provider-url"></param>
    <param nom="jndi-env">java:comp/env</param>
    <param nom="nom-jndi">perso</param>
    <param nom="factory-dao">
        org.jgenea.isolation.defaut.FactoryDAOImpl</param>
    <param nom="factory-connexion-dao">
        org.jgenea.isolation.j2ee.ConnexionFactoryPoolJndiDAOImpl</param>
    <param nom="factory-images-dao">
        org.jgenea.images.registres.FactoryImagesDAOImpl</param>
    <param nom="cache-active">oui</param>
    <param nom="reload-image-pour-affichage">non</param>
    <param nom="repertoire-cache">.</param>
    <param nom="taille-maximum-repertoire-cache">2000000</param>
  </connexion>
</connexions>
```

12.2. Répertoires

Dans le fichier configuration.xml, quatre chemins globaux doivent être spécifiés:

- Le répertoire de base où se trouve les images des sources (clé repertoire-images-sources dans le xml),

- Le répertoire de base où se trouve les images des registres (clé repertoire-images-registres dans le xml),
- Le répertoire de base où il faut générer les images temporaires (clé repertoire-images-temporaires dans le xml),.
- L'url pour accéder au répertoire de base où sont stockées les images temporaires (clé url-images-temporaires dans le xml),.

Les deux derniers sont essentiellement utilisés pour la visualisation des registres numérisés.

Il existe ensuite un chemin spécifique à chaque "fabrique d'images" associée aux connexions. Il est également utilisé pour cette visualisation pour le cache (clé repertoire-cache dans le xml). Elle va de pair avec la taille maximum pour le répertoire de cache en octet (clé taille-maximum-repertoire-cache dans le xml).