Logiciel COLLEC

Installation et configuration

20 octobre 2016

Éric Quinton

IRSTEA - Centre de Bordeaux
50, avenue de Verdun, Gazinet
33612 CESTAS Cedex

Table des matières

Chapitre 1

Le logiciel Collec

Historique

L'unité de recherche *Écosystèmes aquatiques et changements globaux* d'IRSTEA, à Cestas (33), récolte et manipule des échantillons prélevés sur le terrain (ou plutôt, principalement dans l'eau – estuaires, lacs, rivières...), et les stocke, parfois sur des durées très longues : certaines campagnes de collecte ont eu lieu il y a plus de 40 ans.

De plus en plus, des échantillons anciens sont réanalysés (analyses génétiques, étude des ossements des oreilles ou otolithes...), au gré des questions scientifiques à traiter. Le besoin de recourir à un logiciel pour gérer ces matériels est devenu patent.

Dans un premier temps, une rapide recherche a été menée, qui a permis de repérer un certain nombre de logiciels susceptibles de répondre au besoin. Toutefois, leurs limites sont vite apparues : problème de pérennité, ancienneté du code, modèle de distribution insatisfaisant (licence OpenSource obligatoire pour le suivi sur le long terme), résistance aux attaques informatiques, fonctionnalités insuffisantes ou inadaptées au besoin.

Dans un second temps, une étude des besoins réels a été menée. De nouvelles fonctionnalités ont été rajoutées, comme la gestion du stock de matériel utilisé sur le terrain, stocké dans un hangar.

L'unité de recherche s'intégrant au niveau régional avec d'autres organismes, des collaborations avec l'Organisme Aquitain des Sciences de l'Univers (OASU) ou l'université de La Rochelle ont été envisagées. Des échanges productifs ont ainsi pu être mis en place, entre autres sur la gestion de l'étiquetage et le scan des codes-barres.

Le logiciel a largement évolué suite à ces échanges, de nombreuses fonctionnalités ont été rajoutées ou modifiées pour tenir compte des besoin des partenaires potentiels.

Les délais de développement de la première version opérationnelle se sont étalés sur 9 mois, entre la définition des besoins et le développement proprement dit, mené entre juin et octobre 2016. Environ 160 heures auront été nécessaires pour écrire le logiciel. Le code comprend environ 8900 lignes (commentaires compris), dont 3800 concernent l'affichage des pages web.

Principes généraux

Deux types d'objets sont manipulés dans l'application :

— des containers, qui peuvent contenir des objets de tout type : d'autres containers ou des échantillons. Ils peuvent être de différentes nature : site, bâtiment, salle, armoire, caisse, éprouvette...

— des échantillons, qui peuvent être d'un type de container : il y a de nombreux cas où l'échantillon lui-même se confond avec son contenant, par exemple le résultat d'une pêche non triée et stocké dans un bocal.

Un échantillon ou un container sont issus d'un objet unique, qui est doté :

— d'un numéro unique, l'UID, et qui sert de référence dans le logiciel ;

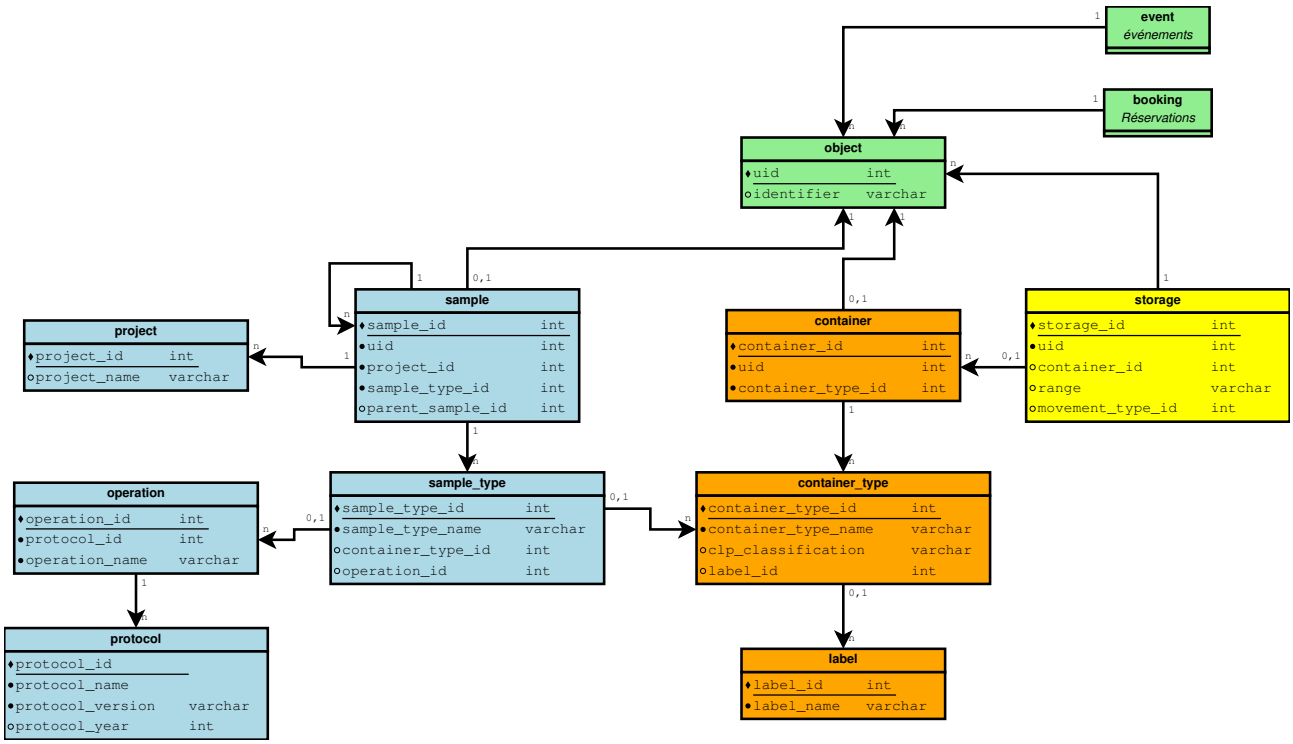

FIGURE 1.1 – Schéma simplifié de la base de données

— d'un identifiant métier, qui servira à le retrouver facilement (le logiciel permet également de définir d'autres identifiants).

Un objet peut être subir un certain nombre d'événements, voire être réservé (fonctionnalité très simplifiée, seul le recouvrement de deux périodes de réservation est signalé).

Tout type de container peut être associé à un modèle d'étiquettes. Les étiquettes peuvent comprendre un code-barre 2D de type QRCode, qui pourra être lu soit à partir d'un terminal dédié (douchette), soit avec un tablette, l'application disposant d'un module capable d'activer la caméra depuis le navigateur et de scanner le code-barre.

Il est ainsi possible d'imprimer des étiquettes tant pour des containers que pour des échantillons, si ceux-ci sont d'un type rattaché à un type de container.

Un échantillon est forcément rattaché à un projet. D'ailleurs, seuls les membres du projet peuvent modifier les informations générales d'un échantillon. Il est également possible de le rattacher à un protocole et à une opération dans le protocole, ce qui permet de suivre son évolution.

Un échantillon peut être subdivisé en d'autres échantillons. Par exemple, un poisson peut se voir extrait des otolithes (os de l'oreille), ce qui permettra de créer 2 nouveaux échantillons d'un autre type, qui pourront toujours être rattachés au poisson parent.

Enfin, dans certains cas de figure, un échantillon peut être composé de plusieurs éléments indifférenciés (plusieurs écailles de poisson prélevées et conservées ensemble). Le logiciel permet d'indiquer les prélèvements et les restitutions réalisées, et affiche le solde (théorique !) restant.

Technologie employée

Base de données

L'application a été conçue pour fonctionner avec Postgresql, en version 9.5. Les versions antérieures peuvent être utilisées, mais seule cette version dispose d'un type de données JSON qui permet de stocker les informations métiers (partie non développée dans la version 1.0).

Langage de développement et framework utilisé

Le logiciel a été écrit en PHP, en s'appuyant sur le framework *Prototypephp*, développé parallèlement par l'auteur du logiciel.

Il a recours à la classe Smarty pour gérer l'affichage des pages HTML. Celles-ci sont générées en utilisant Jquery et divers composants associés. Le rendu général est réalisé avec Bootstrap.

Les étiquettes sont générées en utilisant FOP, une classe Java qui crée des fichiers PDF à partir d'un fichier XML contenant les données et fichier de transformation au format XSL.

Sécurité

L'application a été conçue pour résister aux attaques dites opportunistes selon la nomenclature ASVS v3 de l'OWASP. Des tests d'attaque ont été réalisés en août 2016 avec le logiciel ZapProxy, et n'ont pas détecté de faiblesse particulière.

La gestion des droits est conçue pour :

— qu'un utilisateur, membre d'un projet, ne puisse modifier que les échantillons qui y sont rattachés ;

— que tout utilisateur disposant des droits de gestion peut procéder à une entrée ou une sortie d'un objet ;

— que les responsables d'un projet soient les seuls à pouvoir modifier les paramètres comme les types d'échantillons ou de containers.

L'analyse de sécurité a mis en exergue un besoin de ne pas perdre d'information : si un échantillon est étiqueté et rangé, et que l'information est perdue, il y a de gros risques de ne plus pouvoir l'utiliser ultérieurement. Cela impose la mise en place d'un mécanisme de réplication de la base de données, à implémenter – ou faire implémenter par des administrateurs du système – directement dans Postgresql.

Chapitre 2

Installer le logiciel

Consultez la documentation du framework !

Le logiciel a été conçu à partir du framework *Prototypephp*. La documentation associée Quinton [2016a] récapitule l'ensemble des informations nécessaires pour réaliser l'installation générale (configuration du serveur, définition des droits d'accès, etc.).

De nombreuses reprises figurent ici, mais il n'est pas inutile de se référer au document d'origine. Seuls les exemples dédiés au logiciel sont présentés, mais d'autres informations utiles pourront y être piochées.

Configuration du serveur

La configuration est donnée pour un serveur Linux fonctionnant avec Ubuntu 16.04 LTS Server. Elle peut bien sûr être adaptée à d'autres distributions Linux. Par contre, rien n'a été prévu pour faire fonctionner l'application directement dans une plate-forme windows, même si, en théorie, cela devrait être possible.

Configuration d'Apache

Les modules suivants doivent être activés :

```
a2enmod ssl
a2enmod headers
a2enmod rewrite
```

Modules PHP nécessaires

Modules complémentaires nécessaires :

— *php-mbstring*
— *php-pgsql*
— *php-xdebug* pour les phases de mise au point.

La génération des étiquettes nécessite les paquetages suivants :

— *php-gd* (ou *php5-gd* pour les serveurs en version 5)
— *fop* (qui inclut des bibliothèques java)

Le stockage et l'affichage des photos nécessite :

— *php-imagick* (ou *php5-imagick* pour les serveurs en version 5)

Configuration de l'hôte virtuel et de SSL

L'application ne fonctionne qu'en mode SSL, les cookies de session n'étant pas transmis sur des liens non chiffrés. Voici un exemple de configuration à insérer dans le fichier */etc/apache2/sites-available/default-ssl*

```
<Directory /var/www/html>
    Options FollowSymLinks MultiViews
    AllowOverride all
    Order allow,deny
    allow from all
</Directory>

SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
SSLCipherSuite ECDHE-RSA-AES128-SHA256:AES128-GCM-SHA256:HIGH:!MD5:!aNU
SSLCompression off
```

La chaîne *SSLCipherSuite* est donnée à titre d'exemple, d'autres configurations plus modernes peuvent être implémentées ANSSI [2016].

Pour activer le mode SSL dans Apache :

```
chmod -R g+r /etc/ssl/private
usermod www-data -a -G ssl-cert
a2ensite default-ssl
service apache2 restart
```

Configuration du dossier d'installation

Mécanisme pour faire cohabiter plusieurs instances avec le même code

Il est possible d'utiliser le même code applicatif pour alimenter des bases de données différentes (ou des données stockées dans des schémas différents). Cette fonctionnalité est basée sur l'attribution d'entrées DNS différentes.

Le mécanisme est décrit dans le schéma 2.2.4, page 7.

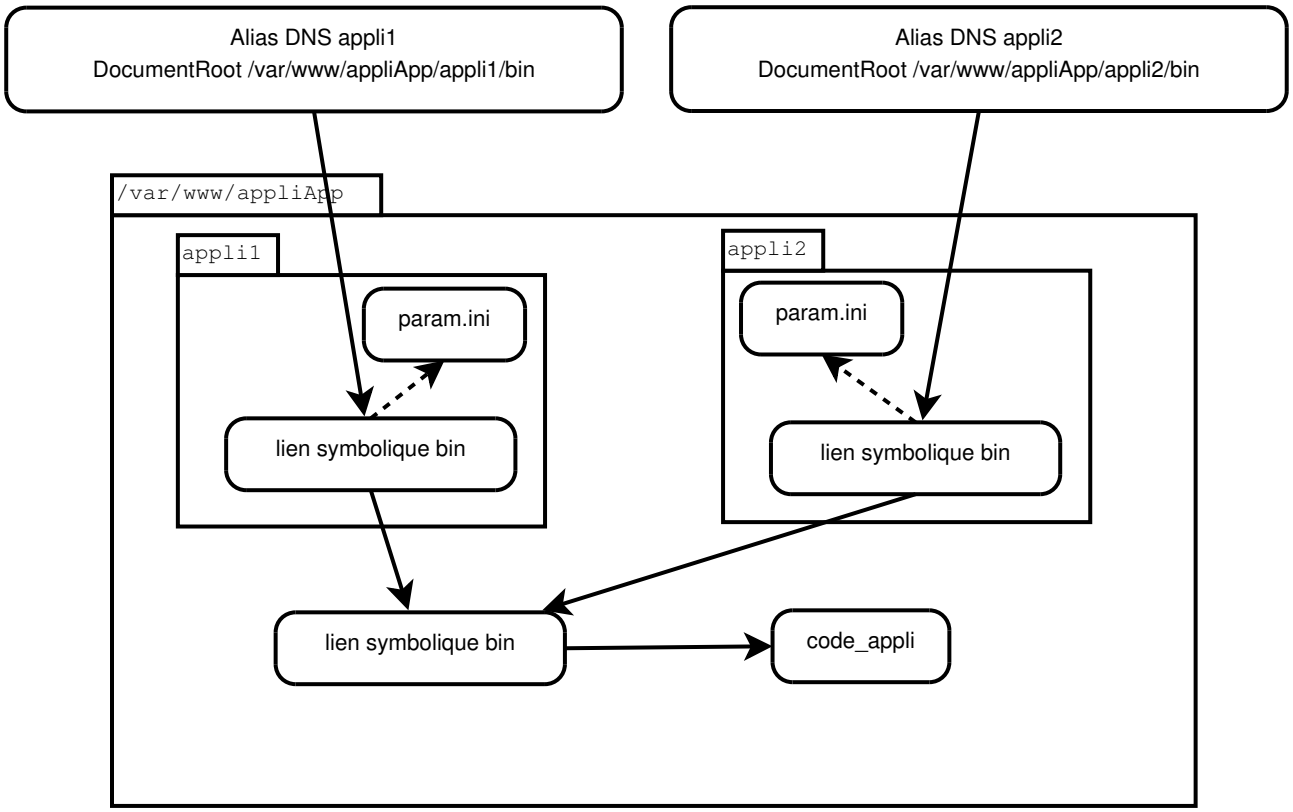

FIGURE 2.1 – Schéma général d'implémentation pour utiliser le même code avec des noms d'application et des jeux de données différents

Dans le paramétrage de l'alias DNS (en principe, dans */etc/apache2/sites-available*), l'application pointe vers le dossier */var/www/appliApp/appli1/bin*. */var/www* correspond à la racine du site web, *appliApp* au dossier racine de l'application, *appli1* au dossier spécifique de l'alias DNS. Ce dossier *appli1* ne contient que deux fichiers : *param.ini*, qui contient les paramètres spécifiques, et *bin*, qui est un lien symbolique vers le dossier *../bin*.

Le dossier *../bin* (donc, dans */var/www/appliApp*) est lui aussi un alias qui pointe vers le code réel de l'application, ici *code_appli*. Le fichier *param.inc.php* décrit l'entrée suivante :

```
$paramIniFile = "../ param . ini ";
```

Le fichier *param.ini* sera cherché dans le dossier parent du code de l'application, c'est à dire soit dans *appli1*, soit dans *appli2* dans cet exemple. Il suffit qu'il contienne les paramètres adéquats pour rendre l'application utilisable dans des contextes différents à partir du même code initial.

Le fichier *param.ini* est le dernier qui est traité par l'application pour récupérer les paramètres.Ceux-ci sont lus dans l'ordre suivant :

param/param.default.inc.php $\rightarrow$ **param/param.inc.php** $\rightarrow$ **../param.ini**

param.ini contiendra les entrées spécifiques liées au DNS utilisé pour accéder à l'application, en principe tout ou partie de celles-ci :

```
APPLI_titre=Gestion des collections EABX
BDD_schema=col , public , gacl
BDD_login=compte_de_connexion
BDD_passwd=mot_de_passe_de_connexion
BDD_dsn=pgsql : host=serveur ; dbname=base_de_donnees ; sslmode=require
GACL_aco=col
APPLI_code=proto
```

Droits à attribuer au serveur web

Le serveur web doit pouvoir accéder en lecture à l'ensemble des fichiers de l'application, et en écriture à deux dossiers :

— *display/templates_c* : fichier utilisé par Smarty pour compiler les modèles de documents HTML ;
— *temp* : dossier de génération des images et des fichiers temporaires

Voici un exemple de script utilisé pour positionner les droits :

```
#!/bin/bash
cp ../param/* param/
chmod -R 770 .
setfacl -R -m u:www-data:rx .
setfacl -R -m d:u:www-data:rx .
mkdir display/templates_c
setfacl -R -m u:www-data:rwx display/templates_c
setfacl -R -m d:u:www-data:rwx display/templates_c
setfacl -R -m u:www-data:rwx temp
setfacl -R -m d:u:www-data:rwx temp
setfacl -R -m d:o::- .
rm -Rf database
rm -Rf test
rm -f param.ini
```

Dans cet exemple, le dossier *../param* contient le fichier *param.inc.php*, qui dispose des paramètres spécifiques à l'implémentation.

Le script est à lancer à la racine du dossier contenant l'application.

Configurer l'application

L'application est configurable par l'intermédiaire de trois fichiers, comme nous venons de le voir :
param/param.default.inc.php → **param/param.inc.php** → **../param.ini**

Le premier fichier contient les paramètres par défaut. Il est systématiquement fourni à chaque nouvelle version de l'application.

Le second est spécifique de l'implémentation. Il comprend notamment les informations liées à la connexion à la base de données, à la méthode d'identification, ou à la recherche des attributs dans l'annuaire LDAP.

le troisième est destiné à traiter la possibilité d'accéder, à partir du même code applicatif, à plusieurs bases de données différentes (*cf.* 2.2.4 *Mécanisme pour faire cohabiter plusieurs instances avec le même code*).

Voici les principaux paramètres utilisés :

Connexion à la base de données

Dans la pratique, deux connexions sont nécessaires : l'une pour accéder à la base des droits, l'autre aux données proprement dites. Voici les paramètres à définir :

Variable	Signification
BDD_login	compte de connexion à la base de données
BDD_passwd	mot de passe associé
BDD_dsn	adresse de la base de données sous forme normalisée

Variable	Signification
BDD_schema	schéma utilisé (plusieurs schémas peuvent être décrits, en les séparant par une virgule - fonctionnement propre à Postgresql)
GACL_dblogin	compte de connexion à la base de données des droits
GACL_dbpasswd	mot de passe associé
GACL_dsn	adresse normalisée
GACL_schema	schéma utilisé
GACL_aco	nom du code de l'application utilisé dans la gestion des droits

TABLE 2.1: Variables utilisées pour paramétrer les connexions

Identification des utilisateurs

Variable	Signification
ident_type	Type d'identification supporté. L'application peut gérer **BDD** (uniquement en base de données),**LDAP** (uniquement à partir d'un annuaire LDAP) **LDAP-BDD** (d'abord identification en annuaire LDAP, puis en base de données), et **CAS** (serveur d'identification *Common Access Service*)
CAS_plugin	Nom du plugin utilisé pour une connexion CAS
CAS_address	Adresse du serveur CAS
CAS_port	Systématiquement 443 (connexion chiffrée)
LDAP	tableau contenant tous les paramètres nécessaires pour une identification LDAP
privateKey	clé privée utilisée pour générer les jetons d'identification
pubKey	clé publique utilisée pour générer les jetons d'identification
tokenIdentityValidity	durée de validité, en secondes, des jetons d'identification

TABLE 2.2: Variables utilisées pour paramétrer l'identification

Configuration de l'accès à l'annuaire LDAP

Les paramètres LDAP sont stockés dans un tableau :

```
$LDAP = array(
                "address"=>"localhost",
                "port" => 389,
                "rdn" => "cn=manager,dc=example,dc=com",
                "basedn" => "ou=people,ou=example,o=societe,c=fr",
                "user_attrib" => "uid",
                "v3" => true,
                "tls" => false,
                "groupSupport"=>true,
                "groupAttrib"=>"supannentiteaffectation",
                "commonNameAttrib"=>"displayname",
                "mailAttrib"=>"mail",
                'attributgroupname' => "cn",
```

```
        'attributloginname' => "memberuid",
        'basedngroup' => 'ou=example,o=societe,c=fr'
);
```

L'application peut non seulement identifier les utilisateurs auprès de l'annuaire LDAP, mais également récupérer les groupes auxquels ils appartiennent dans celui-ci.

Voici les paramètres à indiquer dans ce cas de figure :

Variable	**Signification**
address	adresse de l'annuaire
port	389 en mode non chiffré, 636 en mode chiffré
rdn	compte de connexion, si nécessaire
basedn	base de recherche des utilisateurs
user_attrib	nom du champ contenant le login à tester
v3	toujours à *true*
tls	*true* en mode chiffré
groupSupport	**true** si l'application recherche les groupes d'appartenance du login dans l'annuaire
groupAttrib	Nom de l'attribut contenant la liste des groupes d'appartenance
commonNameAttrib	Nom de l'attribut contenant le nom de l'utilisateur
mailAttrib	Nom de l'attribut contenant l'adresse mail de l'utilisateur
attributgroupname	Attribut contenant le nom du groupe lors de la recherche des groupes (cn par défaut)
attributloginname	attribut contenant les membres d'un groupe
basedngroup	base de recherche des groupes

TABLE 2.3: Variables utilisées pour paramétrer l'accès à l'annuaire LDAP

Paramètres spécifiques

Variable	**Signification**
GACL_aco	nom du code de l'application utilisé dans la gestion des droits (*cf.* 3.1 *Gérer les droits*, page 13)
APPLI_code	Code interne de l'application. **Ce code est essentiel** : il sera inscrit dans les codes-barres générés, pour s'assurer qu'un échantillon est bien issu de la base de données concernée

TABLE 2.4: Variables spécifiques

Créer la base de données

La base de données est composée de deux schémas : l'un pour le stockage des informations d'identification, les droits d'accès, les traces, l'autre pour les données proprement dites.

Le schéma *public* ne devrait jamais être utilisé pour stocker l'information : réservez-le pour les composants communs, comme Postgis si c'est nécessaire.

Les tables de gestion des droits peuvent être communes à plusieurs jeux / applications différentes : la variable *GACL_aco* permet de séparer la gestion des droits pour chaque application, tout en travaillant à partir des mêmes utilisateurs (répartis le cas échéant dans des groupes différents selon le jeu de données considéré).

Les scripts de création des schémas dans la base de données sont stockés dans le dossier *install*.

Créer les tables de gestion des droits

Script à utiliser : *gacl_create.sql*. Les tables nécessaires vont être créées dans le schéma *gacl*, à ne pas modifier.

Le script crée un compte d'administration par défaut :

— login : **admin**

— mot de passe : **password**

Il devra être supprimé quand un autre compte d'administration aura été créé.

D'autre part, les droits par défaut sont positionnés pour le projet *appli* (variable *$GACL_aco* dans les fichiers de paramètres de l'application).

Créer les tables applicatives

Script à utiliser : *col_create.sql*.

Par défaut, le script crée un schéma appelé *col*. Il est possible de créer plusieurs schémas différents, si l'application supporte plusieurs jeux de données (*cf.* 2.2.4 *Mécanisme pour faire cohabiter plusieurs instances avec le même code*, page 6). Dans ce cas de figure, remplacez *col* par le nom du schéma voulu dans les deux premières lignes du script.

Login de connexion

Il est fortement conseillé de créer deux logins de connexion, un pour le schéma des droits, l'autre pour les schémas applicatifs. Ces logins ne doivent pouvoir être utilisés que depuis le serveur web hébergeant l'application.

Cette opération est possible en modifiant le fichier */etc/postgresql/9.5/main/pg_hba.conf* selon ce principe :

```
# Connexions pour les serveurs web
host nom_database userGacl adresse_serveur/32 md5
host nom_database userData adresse_serveur/32 md5
```

et en rechargeant ensuite la configuration de Postgresql avec la commande :

```
service postgresql reload
```

Droits sur les tables

Le compte utilisé pour la connexion au schéma des droits doit pouvoir modifier les informations présentes dans l'ensemble des tables de *gacl*. Il ne doit pas pouvoir accéder aux autres schémas (hormis *public*).

Le compte utilisé pour accéder aux schémas des données doit pouvoir modifier l'ensemble des informations dans les schémas de données, et lire la table *gacl.aclgroup*.

Le plus simple est d'utiliser le logiciel *pgAdmin* pour attribuer les droits.

Scripts de modification

Lors de la livraison de nouvelles versions, il est possible que des scripts de modification soient à exécuter pour mettre à niveau la base de données. Ces scripts doivent être exécutés dans tous les schémas contenant des données applicatives.

Chapitre 3

Administrer l'application

Gérer les droits

Principe général

Les droits sont gérés selon le principe initialement utilisé dans la bibliothèque PHPGACL, aujourd'hui obsolète.

Les logins sont déclarés dans des groupes organisés de manière hiérarchique : un groupe hérite des droits attribués à ses parents.

Les droits utilisés dans le logiciel sont associés à des groupes. Il est possible d'attribuer plusieurs droits à un même groupe, et un droit peut être détenu par des groupes différents.

Si le paramètre *$LDAP["groupSupport"]* est positionné à *true*, les groupes dont fait partie le compte LDAP sont également récupérés, et peuvent être détenteurs de droits dans le logiciel (le nom des groupes est sensible à la casse).

Voici le schéma des tables utilisées pour gérer les droits :

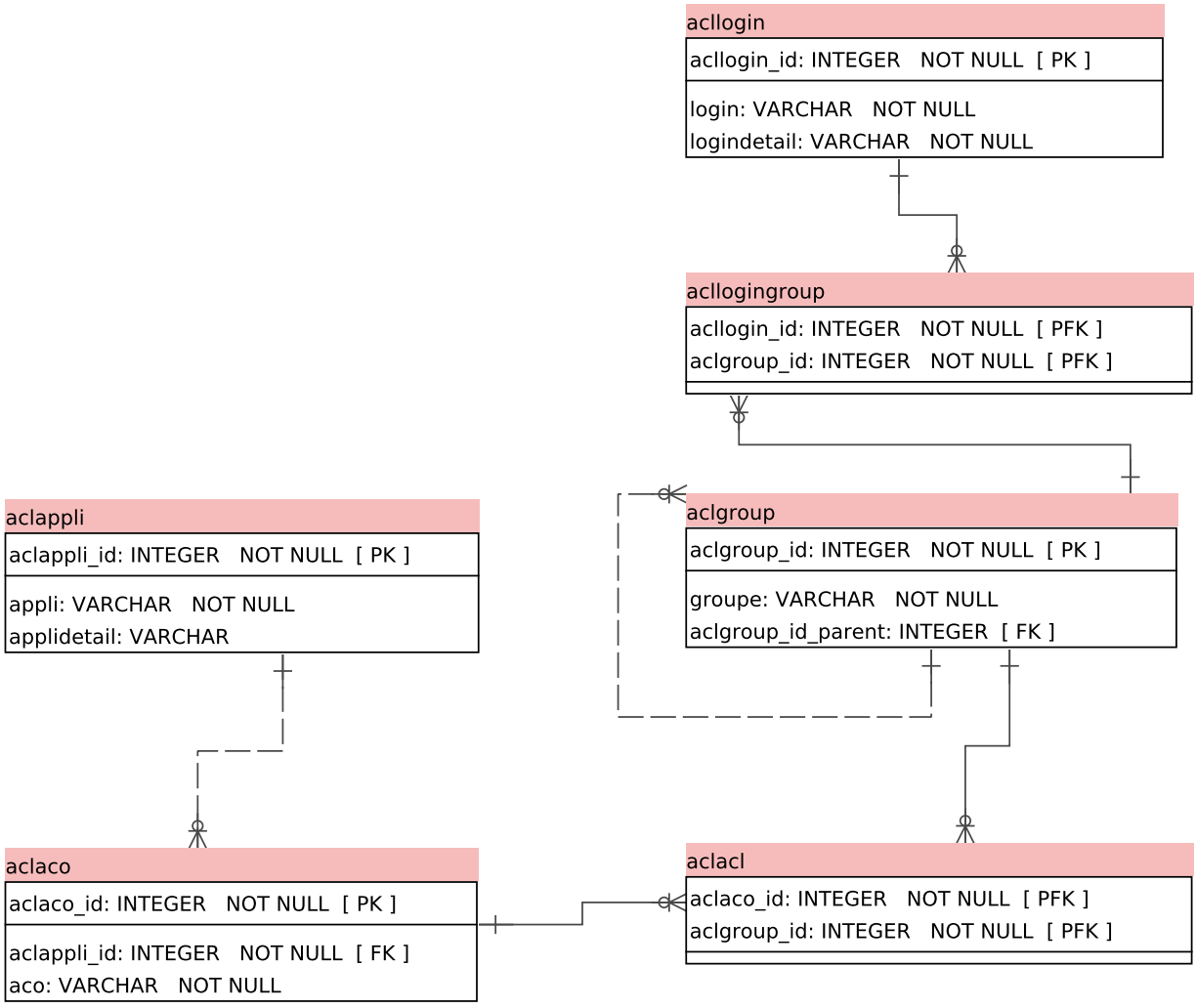

FIGURE 3.1 – Schéma des tables utilisées pour gérer les droits

Voici la description des tables :

acllogin Liste des logins utilisés. Si un compte est créé dans la base locale d'identification, un enregistrement est également créé dans cette table. Pour les identifications LDAP ou CAS, ils doivent être identiques. Si seuls les groupes LDAP sont utilisés pour un compte, il n'a pas besoin d'être décrit ici

aclappli Liste des applications gérées. Il est ainsi possible de gérer, à partir de la même base de données, plusieurs ensembles de droits, qui utilisent les mêmes logins

aclaco liste des droits déclarés dans l'application

aclgroup Liste des groupes contenant les logins, et qui détiennent les droits. Un groupe peut hériter d'un autre groupe. Les droits associés au groupe parent sont également attribués au groupe hérité.

acllogingroup table permettant de déclarer les logins associés à un groupe

aclacl table décrivant les droits détenus par un groupe

Le module d'administration permet de saisir toutes ces informations. Il faut que l'utilisateur dispose du droit *admin*, c'est à dire faire partie du groupe *admin* (configuration par défaut à l'initialisation de la base des droits) pour pouvoir accéder à ces fonctions.

Droits à positionner

Voici les groupes nécessaires pour faire fonctionner correctement l'application :

Droit	**Usage**
admin	Gestion des utilisateurs et des droits
param	Définition des tables de paramètres généraux, Gestion d'un projet
projet	rajout des types d'échantillons ou de conteneurs, import de masse
gestion	Ajout d'un échantillon pour les projets autorisés, entrée/sortie. Droit attribué par défaut si l'utilisateur fait partie d'au moins un projet
consult	Consultation des informations, sans possibilité de modification. Le droit de consultation doit être indiqué volontairement

TABLE 3.1: Liste des droits utilisés

Gestion des projets

Les échantillons doivent être rattachés à un projet, vous devrez en créer au minimum un à partir du menu d'administration. Un utilisateur avec les droits de gestion ne peut modifier que les échantillons pour lesquels il est autorisé (les projets qui sont rattachés au groupe dont il fait partie).

Voici le principe de gestion des droits pour les projets :

Dans *Administration* > *ACL - Groupes de logins*, déclarez les groupes adéquats. En cas d'utilisation des groupes LDAP, les saisir avec la même casse que dans l'annuaire (EABX p. e.).

Il est possible de définir une hiérarchie des groupes, quelle que soit l'origine de l'affectation (base de données ou annuaire Ldap) Dans le cas où l'annuaire Ldap n'est pas utilisé pour gérer les groupes, renseignez les logins en face des groupes dans le même écran.

Dans les projets, sélectionnez les groupes autorisés.

Bibliographie

Eric Quinton. Documentation d'utilisation du framework prototype php, 2016a. URL https://github.com/equinton/prototypephp/blob/bootstrap/database/documentation/prototypephp-documentation.pdf.

ANSSI. Recommandations de securite relatives à tls, 2016. URL http://www.ssi.gouv.fr/uploads/2016/09/guide_tls_v1.1.pdf.

Eric Quinton. Prototypephp, 2016b. URL https://github.com/equinton/prototypephp/tree/bootstrap.