

LABINVENT

(DOCUMENTATION TECHNIQUE)

URL officielle de ce doc : <https://tinyurl.com/labinvent>

Auteurs: E. Pallier, E. Bourrec

Version: 20/12/2018

⇒ **Pour faire une copie** Word, Open Office, ou PDF de ce doc : menu “Fichier/Télécharger...”

*Ce document est davantage une **documentation technique** qu’une documentation destinée aux utilisateurs (à part le chapitre 3).
Il entre dans des détails techniques permettant de comprendre comment est organisé le logiciel LabInvent, comment il fonctionne, comment le configurer, l’adapter et le faire évoluer.*

Pour la documentation (non technique) décrivant l’utilisation du logiciel, cliquer sur le lien ci-dessous :

⇒ [LIEN VERS LA DOC UTILISATEURS](#)

TABLE DES MATIÈRES

1. Installation	3
2. Contenu du projet	4
3. Workflow général	8
3.1. Description générale	8
3.2. Procédure à suivre pour commander un matériel	9
3.3. Diagramme de séquences	10
3.4. Cycle de vie d'un matériel	11
4. Authentification des utilisateurs (avec ou sans LDAP)	13
5. Gestion des profils utilisateurs (Autorisations, ACL)	17
6. LOG	18
7. CONFIGURATION	19
7.1. Configuration des paramètres techniques (BD, mails, debug, ...)	19
7.2. Configuration des paramètres métiers (liés à l'inventaire)	19
8. DEBUG	21
8.1. Debug général	21
8.2. Debug applicatif	22
9. Etiqueteuse	23
10. HOWTO	24
10.1. Version du framework CakePhp utilisé dans le projet	25
10.2. Contrôleur des pages web "simples"	25
10.3. Structure générale d'une page web (TEMPLATE) = default.ctp	25
10.4. Ajouter une nouvelle action sur un contrôleur	26
10.5. Champ facultatif/obligatoire : Comment rendre facultatif ou obligatoire un champ, et définir les vérifications à faire sur ce champ ?	27
10.6. Où définir les éléments passés à une vue (c'est à dire à une action) ?	27
10.7. Adapter LabInvent au laboratoire utilisateur	27

10.8. Ajouter une nouvelle page web	28
10.9. Recherche de matériels	28
10.10. Autres HOWTO plus anciens	29
11. Cycle de développement à respecter	49
12. Documentation Technique (pour les devs)	51
12.1. Modèle de données (Data model)	52

1. Installation

(updated 13/12/18 - EP)

Cette doc est en cours de migration à l'intérieur du présent document.

En attendant, voici le [lien vers la doc d'origine](#)

2. Contenu du projet

(updated 18/12/18 - EP)

Fichier ou Dossier	Description
README.md	Le fichier README général du projet, contient l'historique des versions importantes
TESTS.sh	Script d'exécution des tests (à exécuter régulièrement, après chaque modif)
bin/	Quelques fichiers binaires du framework CakePhp (à utiliser uniquement pendant la phase de développement)
composer.json	Le fichier de description de tous les plugins utilisés par CakePhp et installés automatiquement par Composer
config/	Dossier des fichiers de configuration, notamment app.php qui décrit la configuration de la base de données, des mails, et des tests
database/	Dossier contenant le schéma de base de données pour l'installation ainsi que les scripts de mise à jour à exécuter lorsque la BD a été modifiée (sous-dossier "update")
doc/	Quelques documents, notamment celui-ci
index.php	Le point d'entrée du site web
install/	Le dossier pour l'installation contenant notamment le script général d'installation "installation.sh"
logs/	Les logs de l'application (utiles en pour le debug)
src/	C'est le dossier principal contenant tout le code source de l'application LabInvent pour CakePhp (il aurait aussi pu s'appeler "app"). Voir ci-dessous pour le contenu de ce dossier.
tests/	Le dossier contenant tous les tests exécutés par le script TESTS.sh (voir ci-dessus)
tmp/	Dossier utilisé par LabInvent comme cache des pages web et des tables de la BD

vendor/	Tous les plugins pour CakePhp installés automatiquement par Composer. Lors d'une première récupération du projet, ce dossier est vide. Une fois l'installation faite, ce dossier sera plein de sous-dossiers, un par plugin. Par exemple, vous y trouverez le plugin composer/ lui-même, ainsi que le plugin principal cakephp/ et le plugin phpunit/ qui sert à exécuter les tests (on y trouve même un peu des frameworks symfony et zend...).
webroot/	Le dossier contenant toutes les "ressources" ou éléments utilisés par les pages web, comme les images, les feuilles de style CSS pour la mise en page, les modules javascript pour l'animation des pages, les documents attachés, etc.

Description du dossier principal de l'application src/ :

Controller/	<p>Dans un framework MVC (Modèle-Vue-Contrôleur, ou plutôt ici MTC pour "Template" dans le cas de CakePhp) tel que CakePhp, le Contrôleur (le C du MVC) contient toute la logique métier, en gros presque tout le code de traitement des données. Il récupère une demande d'un utilisateur (qui clique sur une page web), réclame les données nécessaires pour cette demande à la couche Modèle (voir ci-dessous), et les passe à la couche Vue (voir ci-dessous) pour qu'elle les présente à l'utilisateur (une nouvelle page web par exemple). Ce dossier contient donc tous les contrôleurs, un par entité (matériel, utilisateur, fournisseur...), en gros un par table de la BD (mais pas toujours). Par exemple le contrôleur <code>MaterielsController</code> est responsable de traiter les demandes concernant l'entité "matériel". Comme vous pouvez vous en douter, c'est le plus gros contrôleur de <code>LabInvent</code>, étant donné qu'il est centré sur la ressource "matériel". C'est lui qui contient les méthodes CRUD (Create, Read, Update, Delete) permettant de gérer un matériel. Il contient aussi les méthodes permettant de changer le statut d'un matériel pour gérer son cycle de vie (CREATED => VALIDATED => TOBEARCHIVED => ARCHIVED).</p> <p>Tous les contrôleurs sont des classes qui héritent de la classe AppController qui définit un ensemble de comportements par défaut pour tout contrôleur. Notamment, tout comportement général d'un contrôleur doit être déplacé dans <code>AppController</code> plutôt que d'être répété (recopié) dans chaque contrôleur. <code>AppController</code> lui-même hérite de Controller qui est une super-classe de CakePhp offrant elle-même beaucoup de comportements par défaut de tout contrôleur (comme les opérations standard CRUD). Contrairement à <code>AppController</code>, il vaut mieux ne pas modifier la classe <code>Controller</code> car elle risque d'évoluer avec chaque nouvelle version de CakePhp.</p>
Model/ - Behavior/ - Entity/ - Table/	<p>C'est la couche "M" (Model) du pattern MVC (ou plutôt MTC pour "Template") implémenté par le framework CakePhp. Cette couche est responsable de lire ou écrire les données demandées ou rendues par la couche C (Contrôleur), quelque soit le système de gestion de données sous-jacent (MySQL, ou PostgreSQL, ou SQLite, ou simplement des fichiers, etc.). Elle assure ainsi un niveau d'abstraction par rapport à ce système sous-jacent, qui permet de s'en rendre indépendant et même de le remplacer facilement si besoin était. Le niveau d'abstraction est même poussé un peu plus loin en offrant un mapping relationnel-objet (ORM) qui permet de manipuler les données sous forme d'objets plutôt qu'avec des requêtes SQL plus difficiles à écrire et à maintenir. Ainsi, par exemple, <code>\$matériel->save()</code> permettra tout simplement d'enregistrer un objet matériel (représentant une ligne de la table) dans la table <code>matériels</code>, en le créant s'il n'existait pas déjà, ou en le mettant à jour sinon.</p>

Template/
(et View/)

Le dossier Model/ contient notamment les sous-dossiers Entity/ et Table/ :

- le sous-dossier **Entity/** contient un fichier par table qui est une représentation en Php de la structure de la table. Chacun de ces fichiers définit une classe qui hérite de la super-classe **Entity** de CakePhp. Une fois ce fichier écrit, on peut alors générer la table SQL correspondante dans la base de données. Toute modification de la table devrait donc être faite ici plutôt que directement dans la table elle-même. Cela permet la portabilité du projet d'un SGBD à un autre étant donné que cette représentation Php est indépendante de tout formalisme spécifique à un SGBD particulier.
- le sous-dossier **Table/** contient lui aussi un fichier par table, mais décrit plutôt la façon dont les champs (colonnes) de la table doivent être validés. Par exemple, les règles qui permettent de valider un "email" avant de l'enregistrer dans le champ "email" de la table. Chacun de ces fichiers définit une classe qui hérite de la classe **AppTable** (qui elle-même hérite de la super-classe **Table** de CakePhp, à ne pas toucher vu qu'elle peut évoluer avec les nouvelles versions du framework) qui définit quelques règles de validation générales utilisables pour toutes les tables (par exemple, la validation d'un email, ou d'une chaîne de caractères pour définir les caractères interdits par défaut...).

C'est la couche "V" (View) du pattern MVC (ou plutôt MTC pour "Template" ici) implémenté par le framework CakePhp (pour plus d'infos, voir <https://book.cakephp.org/3.0/fr/views.html>). Elle est responsable de présenter les données la plupart du temps sous forme d'une page web. Ces données lui sont passées par la couche (C)ontrôleur. Le dossier View/ (normalement utilisé pour y placer des classes de vue, sous-classes de AppView, elle-même sous-classe de View ; par exemple, PdfView) n'étant pas utilisé pour le moment dans LabInvent, nous ne parlerons que du dossier **Template/**. Celui-ci contient **des sous-dossiers portant chacun le nom du contrôleur qui va l'utiliser**. Les fichiers templates (.ctp) qui sont dans ces sous-dossiers portent un nom correspondant à l'action effectuée. Par exemple, le fichier de vue pour l'action « view() » du contrôleur Products devra normalement se trouver dans **src/Template/Products/view.ctp**.

Un template (.ctp) est un fichier HTML qui peut en plus contenir des instructions particulières dans un format de template propre à CakePhp. Il construit la page web qui sera présentée à l'utilisateur. En général, il y a au moins un fichier par ACTION faite par un contrôleur. Par exemple, les actions CRUD (Create, Read, Update, Delete) sur une table donnée sont permises grâce à des vues dédiées (pages web) telles que (resp.) les templates **add.ctp**, **view.ctp/index.ctp**, et **edit.ctp**, l'action Delete n'ayant en général pas besoin d'une page web pour être exécutée (donc a priori pas de template delete.ctp). Les templates view.ctp et index.ctp représentent l'action Read respectivement sur UNE ligne de la table ou sur TOUTES les lignes. Ensuite, on peut avoir d'autres templates selon les besoins, tels que **find.ctp** par exemple pour afficher une page web de recherche des matériels.

Dans le dossier **Template/**, on trouve quelques sous-dossiers particuliers :

- **Element/** : contient notamment des parties (blocs) de pages web réutilisés dans différentes pages
- **Error/** : contient les pages web qui sont affichées en cas d'erreur
- **Fichemetrologiques/, Formules/, et Unites/** : contiennent toutes les vues spécifiques pour le module Métrologie
- **Layout/** : contient notamment la page web **default.ctp** qui sert de page de base (template) à toutes les pages web du site ; c'est elle qui donne la structure générale des pages du site (menu à gauche, header, footer, ...) ; cette page contient aussi le numéro de version et la date de LabInvent qui doit être mise à jour à chaque modification du projet
- **Pages/** : contient toutes les "pages" web simples (plutôt "statiques") du site, comme la page **"home.ctp"** (page

d'accueil), la page "**about.ctp**", la page "**tools.ctp**" (menu Outils), la page "**printers.ctp**" (présentation de l'étiqueteuse), ou la page **infos.ctp** (informations systèmes concernant le système hôte, accessible uniquement au profil SuperAdmin)

- **QrCodes** / : page `creer.ctp` permettant la création du QrCode associé à un matériel, sur chaque page web de visualisation d'une fiche matériel

La couche vue de CakePHP peut être constituée d'un certain nombre de parties différentes. Chaque partie a différents usages qui seront présentés dans ce chapitre :

- **templates**: Les templates sont la partie de la page qui est unique pour l'action lancée. Elles sont la substance de la réponse de votre application.
- **elements** : morceaux de code de view plus petits, réutilisables. Les elements sont habituellement rendus dans les vues.
- **layouts** : fichiers de template contenant le code de présentation qui se retrouve dans plusieurs interfaces de votre application. La plupart des vues sont rendues à l'intérieur d'un layout.
- **helpers** : ces classes encapsulent la logique de vue qui est requise à de nombreux endroits de la couche view. Parmi d'autres choses, les helpers de CakePHP peuvent vous aider à créer des formulaires, des fonctionnalités AJAX, à paginer les données du model ou à délivrer des flux RSS.
- **cells**: Ces classes fournissent des fonctionnalités de type controller en miniature pour créer des composants avec une UI indépendante. Regardez la documentation [View Cells](#) pour plus d'informations.

3. Workflow général

3.1. Description générale

(updated 12/12/18 - Elodie Bourrec)

Les matériels figurent dans Labinvent s'ils font plus de 800 E, ou s'il y a un intérêt technique à inventorier ce type de matériel.

Un usager veut commander un matériel :

Il va créer une fiche (avec toutes les caractéristiques et description appropriée) et avoir un numéro d'inventaire. Le matériel a un statut "Created". Il peut éventuellement être supprimé.

Avec ce numéro (ou la fiche) il envoie sa demande (avec le devis) à son personnel de gestion qui va émettre le bon de commande, et éventuellement, sur Labinvent, remplir la partie administrative du matériel en question.

Quand le matériel arrive :

Avec le bon de livraison, le matériel peut être validé (c'est là que certaines questions d'organisation, et de droit sur l'appli, se posent) Les gestionnaires peuvent valider les matériels mais souvent, ils n'en voient que le Bon de livraison. Pour compléter la fiche matériel, avec num série, date de garantie ... il va falloir que les personnes qui l'ont commandé se reconnectent.

Une fois le matériel "validated", il ne peut plus être supprimé. Il ne peut qu'être sorti de l'inventaire.

Tous les personnels du labo peuvent demander une sortie d'inventaire, il n'y a que les gestionnaires (profil "administration") qui peuvent faire la sortie effective.

Les matériels restent en base avec le statut "Archived" mais ne sont plus visibles sur les listes de l'appli. Les gestionnaires peuvent voir la liste du matériel archivé.

Il y a plusieurs profils dans l'appli :

User - utilisateur : Mr tout le monde. Il voit tous les matériels mais ne peut modifier que les siens.

Responsable : nous avons pensé que les responsables de groupe pouvaient être valideurs et avoir accès en modification à tous les matériels de leur groupe, mais chez nous, nous ne nous en servons pas. Il faudrait voir avec les autres labos s'ils se servent de ce profil et s'il est fonctionnel.

Administration : profil pour les gestionnaires. La gestion voit tous les matériels et peut tout modifier. Ils ont accès en plus à la partie administrative (bon de commande, EOTP, ...)

Super-administrateur: peut tout faire sauf sortir les matériels de l'inventaire.

3.2. Procédure à suivre pour commander un matériel

(updated 17/12/18 - Etienne Pallier)

Voici le message d'accueil affiché à la connexion d'un utilisateur de LabInvent ayant le profil "utilisateur" (c'est à dire le profil de plus bas niveau, correspondant à un utilisateur lambda, "non privilégié"). Ce message lui indique la marche à suivre (le "workflow" à respecter) pour commander un matériel. Les **acteurs** (le demandeur et le gestionnaire) sont en jaune, les **actions** en rouge.

*Voici la procédure (en 8 étapes) pour passer commande d'un matériel de plus de 800€ (matériel **inventoriable**) :*

*(je **peux** aussi, si je le désire, suivre cette procédure pour un matériel < 800€, afin qu'il soit référencé)*

*(je deviens "**demandeur/utilisateur/référent**" de ce matériel)*

1. **J'obtiens un devis**
2. **Je crée une fiche matériel** (clic sur "Nouveau Matériel") avec les quelques informations obligatoires (notamment une **description précise** du matériel) (éventuellement, je peux y associer le devis en document attaché)
3. **J'imprime ma fiche et l'amène (avec le devis) à un gestionnaire** (ou bien je l'envoie par email)
4. **Le gestionnaire** retrouve cette fiche (en tapant son n° interne labo dans le champ "Recherche") et la **complète** avec les infos administratives
5. **Le gestionnaire crée le bon de commande** pour ce matériel et **passé la commande** (éventuellement, il peut associer le bon de commande à la fiche matériel en document attaché)

...ATTENTE DE LA LIVRAISON...

6. **A la livraison** du matériel, **le gestionnaire valide la fiche matériel** (éventuellement, il peut y associer le bon de livraison en document attaché)
=> (je reçois un mail qui m'informe de l'arrivée du matériel et me demande de **vérifier** ma fiche)
7. **Le gestionnaire imprime l'étiquette** d'inventaire associée au matériel **ainsi que la fiche complète** du matériel et **la joint au carton du matériel**
8. **Je viens chercher mon nouveau matériel** et y **colle l'étiquette** d'inventaire

3.4. Cycle de vie d'un matériel

Le statut d'un matériel change selon le workflow suivant :

- 1) Un utilisateur lambda le crée (n'importe qui du labo) --> **CREATED**
- 2) L'Administration le valide (après avoir éventuellement complété la fiche) --> **VALIDATED**
- 3) Un utilisateur lambda demande à l'archiver --> **TOBEARCHIVED**
- 4) L'Administration le sort de l'inventaire --> **ARCHIVED**

Notes :

- Dans l'idéal, le matériel est d'abord créé par l'utilisateur concerné, puis mis à jour par l'administration au moment de la commande (puis validé)
- L'administration peut toujours retrograder le statut d'un matériel (ce qui revient à annuler un changement de statut)

Créer un matériel ==> passe alors en statut **CREATED** ==> peut alors être éventuellement supprimé

Valider un matériel **CREATED** ==> passe alors en statut **VALIDATED** => ne peut plus être supprimé

Demander l'Archivage d'un matériel **VALIDATED** ==> passe alors en statut **TOBEARCHIVED**

Archiver (sortir de l'inventaire, en validant une demande d'archivage d'un matériel **TOBEARCHIVED**) ==> statut **ARCHIVED** (n'est alors plus visible, sauf pour admin)

Désarchiver un matériel ==> repasse de **TOBEARCHIVED** ou **ARCHIVED** à **VALIDATED**

En résumé :

CREATED (fiche matériel créée) ==> **VALIDATED** (= matériel livré [et vérifié]) ==> **TOBEARCHIVED** ==> **ARCHIVED**

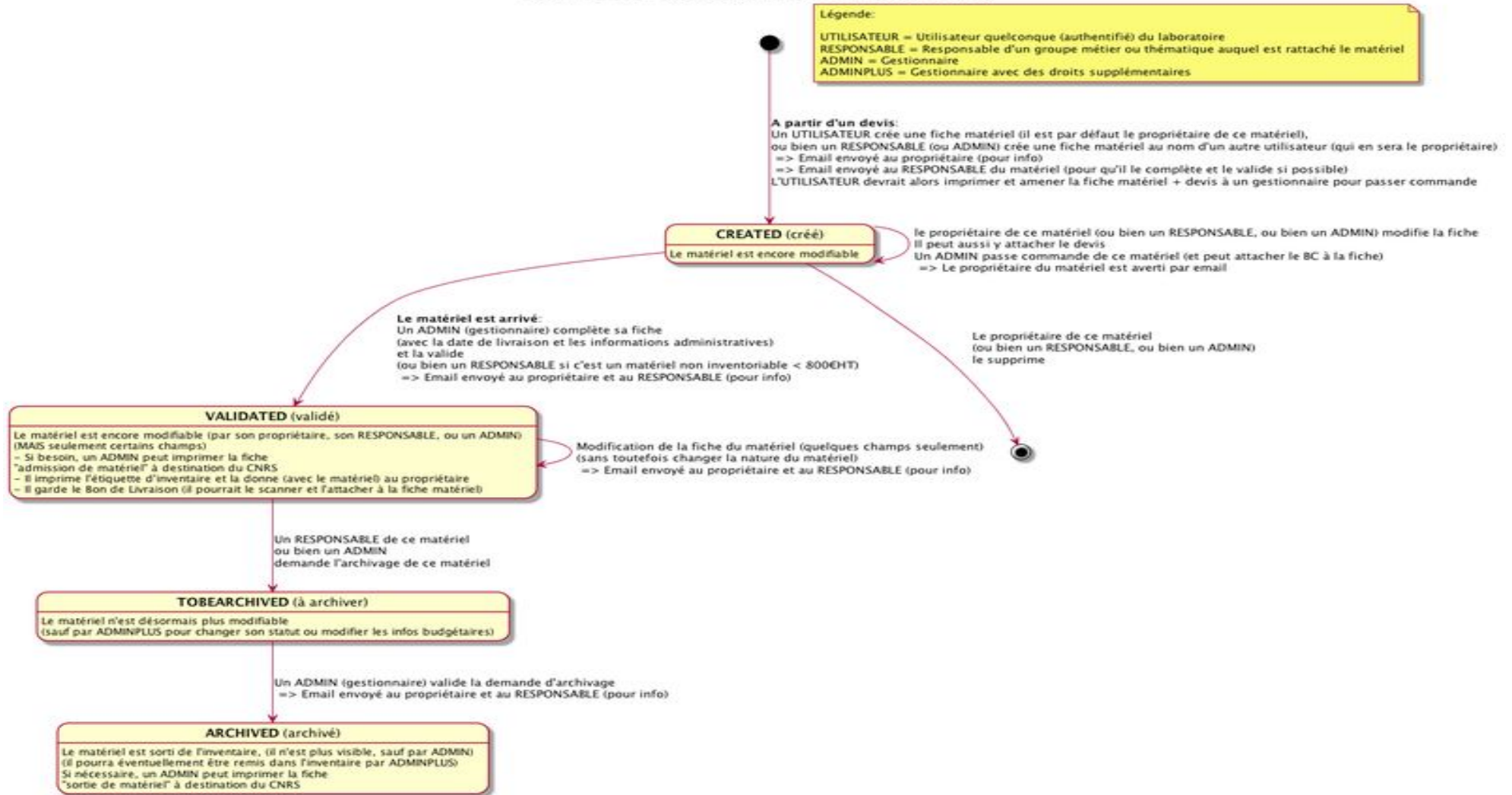
||
||
v

DELETED

Attention : **VALIDATED** = matériel livré [+ fiche vérifiée] (dans l'idéal)

Diagramme états-transitions :

CYCLE DE VIE D'UN MATÉRIEL (SES DIFFÉRENTS ÉTATS)



4. Authentification des utilisateurs (avec ou sans LDAP)

(updated 17/12/18 - EP)

Nous décrivons ici le processus de CONNEXION d'un utilisateur avec un login et mot de passe. Il est possible avec ou sans LDAP.

Ce processus se nomme en anglais “**authentication**”. Pour plus d'information à ce sujet, lire la documentation de CakePhp (<https://book.cakephp.org/3.0/en/controllers/components/authentication.html>)

1) Chemin parcouru (dans l'ordre) depuis le tout début, avant d'arriver à la page de login

(Cette étape est décrite pour être exhaustif, mais vous pouvez aller directement à l'étape suivante, “2- LOGIN”)

webroot/index.php

config/bootstrap.php => lit config/app.php

src/Controller/PagesController/display()

src/Controller/UsersController/login() sans POST

src/Template/Users/login.ctp => formulaire de login => POST

2) LOGIN (avec ou sans LDAP)

Une fois le formulaire de LOGIN “posté” (validé par l'utilisateur), on arrive à l'étape de login proprement dite

Pour info, si ça bogue (par exemple, à l'époque d'un fameux stagiaire..., on retournait "YOLO" au lieu de FALSE en cas d'échec de connection ldap, et ça plantait lamentablement, je m'en souviens très très bien...),
⇒ c'est **src/Template/Error/error400.ctp** qui prend le relais...

Voici la description du processus de login (authentification) :

a) *src/Controller/UsersController.php*, fonction **login()** :

```
$user = $this->LdapAuth->connection();
```

b) *src/Controller/Component/LdapAuthComponent.php*, fonction **connection()** :

```
// Get login and password entered by the current user (who is trying to connect)
```

```
$login = $this->request->getData('ldap');
```

```
$password = $this->request->getData('password');
```

```
$return TableRegistry::get('LdapConnections')->ldapAuthentication($login, $password);
```

c) *src/Model/Table/LdapConnectionsTable.php*, fonction **ldapAuthentication(\$user_login, \$user_password)** :

```
if ($this->USE_LDAP) {           // We are using LDAP
    $LDAP_ANONYMOUS = true;
    if (strlen(trim($user_password)) == 0) return FALSE;           // No connexion allowed without password

    // (1) Set LDAP parameters
    $just_these = [];           // By default, get all fields
    if ($LDAP_ANONYMOUS) {           // - Anonymous connection (IRAP, IAS, LATMOS)
        //$dn = $this->baseDn; // "ou=users,dc=irap,dc=omp,dc=eu"
        $auth_dn = ""; //!= $this->authDn;
        $binddn = $this->authenticationType . '=' . $user_login;
        $ldappass = $user_password;
        $filter = '(!.$binddn.)';
    }
    else {           // - Authenticated connection (CRAL)
        //$dn = $this->baseDn; // "dc=univ-lyon1,dc=fr";
        $auth_dn = "CN=svc_ldap_cral,OU=users,OU=27,OU=sim,OU=univ-lyon1"; //!= $this->authDn;
        $binddn = $auth_dn;
        $ldappass = "lemotdepasse";
        $filter = "(&(objectClass=person)(memberOf:1.2.840.113556.1.4.1941:=cn=ucbl.osu.cral,ou=groups,ou=27,ou=sim,ou=univ-lyon1,dc=univ-lyon1,dc=fr))";
    }
    $binddn .= ',$this->baseDn;

    // (2) Connection
    $ldapConnection = ldap_connect($this->host, $this->port) or die("Could not connect to $this->host (port $this->port)");
    if ($ldapConnection) {
        ldap_set_option($ldapConnection, LDAP_OPT_PROTOCOL_VERSION, 3);
        $ldapbind = ldap_bind($ldapConnection, $binddn, $ldappass) or die( "Could not bind to LDAP server.". ldap_error($ldapConnection) );
        if ($ldapbind) {
            $search = $this->getUserAttributes($user_login, $ldapConnection, $LDAP_ANONYMOUS, $filter, $just_these);
            if ($search === false) die("Could not get user attributes from LDAP server, response was: " . ldap_error($ldapConnection) );
            return $search[0];
        }
    }
}
```


d) *src/Model/Table/LdapConnectionsTable.php*, fonction **getUserAttributes**(\$userName, \$ldapConnection="", \$LDAP_ANONYMOUS=true, \$filter="", \$just_these=[]) :

```
$results = ldap_search($ldapConnection, $this->baseDn, $filter, $just_these) or die("Could not search to LDAP server response was: " . ldap_error($ldapconn) );
$info = ldap_get_entries($ldapConnection, $results);
//echo $info["count"]." entries returned\n";
return $info;
```

e) Retour à *src/Controller/UsersController.php*, fonction **login**() :

```
// Juste après : $user = $this->LdapAuth->connection();
if ($user != FALSE) {
    $this->LdapAuth->setUser($user);
    return $this->redirect($this->LdapAuth->redirectUrl());
}
```

5. Gestion des profils utilisateurs (Autorisations, ACL)

Cette partie fait l'objet d'un [document séparé](#)

6. LOG

(updated 19/12/18 - EP)

Des messages de debug et d'erreur peuvent être loggés grâce au système de logging de CakePhp qui est configuré dans le fichier de configuration générale **config/app.php** :

```
/**
 * Configures logging options
 */
'Log' => [
    'debug' => [
        'className' => 'Cake\Log\Engine\FileLog',
        'path' => LOGS,
        'file' => 'debug',
        'levels' => ['notice', 'info', 'debug'],
        'url' => env('LOG_DEBUG_URL', null),
    ],
    'error' => [
        'className' => 'Cake\Log\Engine\FileLog',
        'path' => LOGS,
        'file' => 'error',
        'levels' => ['warning', 'error', 'critical', 'alert', 'emergency'],
        'url' => env('LOG_ERROR_URL', null),
    ],
],
```

Remarque :

```
error_log("bla bla bla");
==> écrit dans logs/error.log
```

Dans une ancienne version de LabInvent, on pouvait faire ceci (il faudrait remettre ça en place) :

```
$this->log('Something broke');
==> écrit dans cakephp/app/tmp/logs/labinvent.log
```

7. CONFIGURATION

(updated 20/12/18 - EP)

Il y a 2 niveaux de configuration du logiciel.

7.1. Configuration des paramètres techniques (BD, mails, debug, ...)

Cette configuration se fait via la modification du fichier Php qui est “**config/app.php**”.

Pour lire un paramètre de configuration dans le code (dynamiquement) :

```
// Lecture de la valeur du paramètre “debug”  
if ( Configure::read('debug') ) ...
```

On pourrait ajouter d'autres paramètres dans ce fichier, selon le besoin, mais pour cela, on préférera la solution suivante (via la BD).

7.2. Configuration des paramètres métiers (liés à l'inventaire)

Cette configuration se fait directement **via des pages web** qui interagissent avec une **table de la base de données** nommée “**configurations**”.

Pour cela, aller dans le menu Outils, puis cliquer sur les liens suivants :

- **Configuration générale** de l'application : pour configurer le LDAP, les mails, le nom du labo, etc...
- **Gérer le contenu variable** de l'application : pour gérer les catégories de matériels, les sites, les organismes, les types de suivis, les groupes thématiques et métier, les types de docs, les fournisseurs, les unités et les formules (module métrologie), ...
- **Gérer les utilisateurs privilégiés** : pour gérer les utilisateurs privilégiés (ajouter, modifier, supprimer)

Pour lire un paramètre de configuration dans le code (dynamiquement) :

- **dans un contrôleur** (ex: src/Controller/Pages/PagesController.php) :
 - // La variable \$this->**confLabinvent** est **définie** dans la classe mère **AppController** (pour que tous les contrôleurs en héritent), dans la fonction **initialize()**, comme ceci :

- ```
$this->confLabinvent = TableRegistry::get('Configurations')->find()->first();
```
- // Elle est donc utilisable dans n'importe quel contrôleur  
// ex: pour savoir si on est en mode INSTALL ou pas :  
\$configuration = \$this->**confLabinvent**  
if (\$configuration->mode\_install) ...
  - // Elle est aussi **passée à TOUTES les vues** (templates) via la fonction **beforeRender()** de ApplicationController, sous la forme d'une variable nommée **\$configuration** :  
\$configuration = \$this->confLabinvent;  
\$this->**set('configuration'**, \$configuration);
  - **dans un template** (ex: src/Template/Pages/tools\_sm.ctp), la variable \$configuration (passée à toutes les vues par ApplicationController comme expliqué ci-dessus) est disponible :  
// ex: pour savoir si la variable de configuration "metrologie" est à True ou False :  
if (\$configuration->metrologie) ...

Pour ajouter un nouveau paramètre de configuration :  
**(TODO:)**

# 8. DEBUG

*(updated 19/12/18 - EP)*

CORRIGER UNE ERREUR, RESOUDRE UN PROBLEME, COMMENT DEBOGUER (DEBUG)...

Il y a 2 niveaux de debug dans LabInvent.

## 8.1. Debug général

### COMMENT VOIR LES REQUETES SQL FAITES PAR CAKEPHP ?

Vous devez d'abord vous assurer que vous êtes bien en mode **DEBUG=true** dans le fichier de configuration config/app.php.

Dans le code, on peut aussi lire la variable de configuration "debug" ainsi :  
if ( Configure::read('debug') ) ...

Ensuite, dans le layout general (src/Template/Layout/default.php), ajouter cette ligne :  
<?php echo \$this->element('SQL\_DUMP'); ?>

### Comment pister les erreurs ?

Cakephp vous affiche une erreur, mais vous ne savez pas d'ou vient le probleme.  
Don't panic.  
En general, un bon moyen de le savoir est de lire le fichier de log des erreurs  
dans **logs/error.log**

Dans le dossier **logs/**, vous trouverez d'autres logs utiles :  
- **labinvent.log**  
- **debug.log** (pas sur que ce fichier soit encore utilisé...)

Remarque :

```
error_log("bla bla bla");
==> écrit dans logs/error.log
```

## 8.2. Debug applicatif

C'est un mode "debug" pour l'application LabInvent qui se met alors à afficher des informations techniques en haut de page (header).  
Pour y passer, il suffit d'aller dans le menu **Outils** puis "**Passer en mode DEBUG**".  
Pour revenir au mode normal, menu **Outils** puis "**Stopper le mode DEBUG**".

# 9. Etiqueteuse

*(updated 19/12/18 - EP)*

<https://projects.irap.omp.eu/projects/inventirap/wiki/Installation#I-Etiquettes-optionnel>



# 10. HOWTO

Ce document est un HOWTO, c'est à dire un guide technique pour savoir comment faire quoi.

Il donne des solutions à différents types de besoins ou problèmes, et explique aussi où trouver quoi.

## 10.1. Version du framework CakePhp utilisé dans le projet

L'information est dans le fichier vendor/cakephp/cakephp/**VERSION.txt**

Ou bien, exécuter cette ligne php :

```
Configure::version(); // 3.5.11
```

## 10.2. Contrôleur des pages web "simples"

Controleur des pages simples = Controller/PagesController.php)

Page d'accueil = src/Template/Pages/home.ctp

## 10.3. Structure générale d'une page web (TEMPLATE) = default.ctp

src/Template/Layouts/default.ctp contient la structure suivante :

```
<div id="container">
 <div id="header">
 LE HEADER AVEC SON LOGO
 <div class="user">
 BIENVENUE $userName (ou invité)
 </div>
 </div>
 <div id="content">
 <?php echo $this->Session->flash(); ?>
 <?php echo $this->fetch('content'); ?>
 </div>
 <div id="footer">
 LE FOOTER
 </div>
</div>
```

La DIV "content" insère 2 contenus :

- le message flash éventuel (qui dit si une opération demandée s'est bien passée ou pas...)
- la section "content", qui n'est autre que le coeur de la page

La page d'ACCUEIL n'est qu'un "content" parmi d'autres.

Elle se trouve dans src/Template/Pages/home.ctp

(elle est affichée par le contrôleur de pages nommé PagesController)

#### 10.4. Ajouter une nouvelle action sur un contrôleur

Ex: ajout de l'action `statusCreated()` dans le contrôleur `Matériel`

Il faut l'ajouter à 2 endroits dans `MaterielsController.php` :

a) ajouter une méthode `statusCreated()` :

```
public function statusCreated($id = null, $from = 'index') { ... }
```

b) ajouter un cas 'statusCreated' dans la méthode `isAuthorized()` qui doit retourner `true` pour autoriser cette action :

```
case 'statusCreated': ... return true; ...
```

La voici en détail :

```
case 'statusCreated': // de-validation d'un matériel (re passe à CREATED)
 $id = (int) $this->request->getParam('pass.0');
 // Admin + ok
 if ($this->USER_IS_ADMIN_AT_LEAST()) return true;
 // Resp ok if same group
 if ($this->USER_IS_RESP() && $this->isRespGroup($id, $user[$ldapAuthType])) return true;
 // User not ok
 break;
```

## 10.5. Champ facultatif/obligatoire : Comment rendre facultatif ou obligatoire un champ, et définir les vérifications à faire sur ce champ ?

=> src/Model/Table/<Model>Table.php

ex : src/Model/Table/MaterielsTable.php pour définir les règles sur les champs de la table "materiels"

## 10.6. Où définir les éléments passés à une vue (c'est à dire à une action) ?

=> src/Controller/<Model>Controller/nom\_de\_la\_vue\_ou\_action()

ex : src/Controller/MaterielsController/edit() pour définir (avec set()) les éléments passés à la vue "edit" des matériels

## 10.7. Adapter LabInvent au laboratoire utilisateur

(updated 19/12/18 - EP) ⇒ **A compléter**

### Vues (Pages) concernées :

- src/Template/Layout/default.ctp (template)
- src/Template/Pages/home.ctp (Accueil)
- src/Template/Pages/about.ctp (A propos)
- src/Template/Pages/printers (Etiqueteuses)

### Contrôleurs concernées :

- src/Controller/MaterielsController.php (fonction getLabNumber())

### Documents concernés :

- Etiquette (**TODO**)
- src/Template/Documents/admission.ctp (Document d'admission UPS et CNRS)

## 10.8. Ajouter une nouvelle page web

*(updated 19/12/18 - EP)*

Par exemple, voici ce que j'ai fait pour ajouter la page "about", qui est accessible via l'url <https://inventirap.irap.omp.eu/pages/about>

1) Aller dans src/Template/Pages/

1) Faire un copier/coller d'une page existante, par exemple infos.ctp, et l'appeler about.ctp

2) Remplir cette page avec le contenu souhaité

3) Tester que cette page est bien accessible via l'url <http://.../pages/about>

4) Ajouter un lien vers cette page, soit dans le menu outils (src/Template/Pages/tools.ctp), soit dans le menu général (src/Template/Element/menu.ctp)

5) Si cette page ne doit pas être accessible à tout le monde, définir le profil minimum exigé dans src/Controller/PagesController.php, dans la fonction display() :

```
if ($page == about) {
 // Autoriser seulement à partir du role ADMIN
 if (! $this->USER_IS_ADMIN_AT_LEAST()) return $this->redirect('/');
}
```

(ou bien ajouter une ligne dans le tableau \$minProfileAllowedForPage)

## 10.9. Recherche de matériels

*(updated 19/12/18 - EP)*

3 methodes :

- Recherche générale (champ "Recherche" sous le menu général à gauche) : src/Template/Element/menu.ctp

⇒ appelle l'action "materiels/find"

- VUE de recherche : src/Template/Materiels/find.ctp

- ACTION de recherche : src/Controller/MaterielsController (methode find())

## 10.10. Autres HOWTO plus anciens

Le contenu de ce chapitre est à utiliser avec précaution car, étant assez ancien, il risque de ne pas être complètement utilisable (bien qu'il puisse encore l'être dans la plupart des cas). Cependant il reste utile pour information.

### INSTALLATION

\*\*\*\*\*

La procedure d'installation est decrite dans le fichier INSTALLATION.txt qui se trouve dans le dossier install/  
Pour une installation à partir d'Eclipse, voir le document install/manual\_install/INSTALLATION\_MANUELLE\_mode\_expert.txt

### OU EST QUOI (WHERE IS WHAT) ?

\*\*\*\*\*

- OU mettre a jour la VERSION du soft (AVANT CHAQUE COMMIT) ?  
En attendant mieux, on fait cela a la fin du fichier cakephp/app/View/Layouts/default.ctp
- OU EST LA PAGE GENERALE (qui contient tous les elements) ? : cakephp/app/View/Layouts/default.ctp
- OU EST LA PAGE D'ACCUEIL ? : app/View/Pages/home.ctp
- OU SONT LES MENUS ? : app/View/Elements/
  - menu general + Recherche generale : menu.ctp
  - sous le menu general, et sous le champ "Recherche", titre du modele a ajouter : menu\_index.ctp
- OU EST LA FEUILLE DE STYLE CSS ? : cakephp/app/webroot/css/inventirap.css
- OU EST DEFINIE LA PAGINATION ?  
Dans le Controleur  
Ex : la pagination des materiels est definie dans app/Controller/MaterielsController.php

```
public $paginate = array(
 'limit' => 50,
 'order' => array('Materiel.id' => 'desc');
```

- OU SONT LES INFOS CONCERNANT UNE ENTITE quelconque (Materiel, Emprunt, Suivi, Utilisateur) ?  
par exemple, ou trouver les infos sur l'entite "Materiel" ? : Aller dans cakephp/app/
- le Modele : Model/Materiel.php
- le Controleur : Controller/MaterielsController.php
- les Vues (templates) : View/Materiels
  - Vue de consultation : scaffold.view.ctp
  - Vue d'ajout (add) et edition (edit) : scaffold.form.ctp
  - vue de liste : index.ctp

- OU SONT DEFINIS LES ROLES (ACL) ?

Dans app/Model/Utilisateur.php :

```
private $acceptedRoles = array ('Utilisateur', 'Responsable', 'Administration', 'Super Administrateur');
public function getAuthenticationLevelFromRole($role) {
 if ($role == 'Utilisateur')
 return 1;
 elseif ($role == 'Responsable')
 return 2;
 elseif ($role == 'Administration')
 return 3;
 elseif ($role == 'Super Administrateur')
 return 4;
 return 0;
}
```

ANCIEN FICHER DE CONFIG labinvent.php

\*\*\*\*\*

*(updated 19/12/18 - EP)*

(cf <http://book.cakephp.org/2.0/fr/development/configuration.html#loading-configuration-files>)

Ce fichier n'est plus utilisé car maintenant la configuration est mise dans une table "configurations".

Je laisse quand même cette section pour information sur la manière de gérer la configuration via un fichier php.

1) J'ai créé le nouveau fichier de config dans app/Config/ (par exemple labinvent.php)

Ce fichier doit au minimum contenir un tableau nommé \$config :

```
$config = array(
 'labName' => 'IRAP',
 ...
);
```

2) Charger ce nouveau fichier de config au démarrage

Pour cela, ajouter cette ligne dans app/Config/bootstrap.php :

```
Configure::load('labinvent');
```

3) Lire (ou même modifier) les paramètres de ce fichier de config, depuis N'IMPORTE OU (contrôleur, modèle, vue) :

```
debug(Configure::version()); // pour afficher la version de Cakephp
debug(Configure::read()); // tout lire
debug(Configure::read('localisation')); // le tableau $localisation
$labName = strtoupper(Configure::read('localisation.labNameShort'));
debug(Configure::read('localisation.labName'));
if (Configure::read('USE_LDAP')) ...
if (Configure::read('debug') > 0) ...
```

On peut même modifier la valeur d'un paramètre dynamiquement comme ceci :

```
Configure::write('debug',2)
```

C'est d'ailleurs ce qui est fait dans app/Config/core.php

4) Penser à modifier le script d'installation install/installation.sh pour qu'il prenne en compte ce nouveau fichier

## GOOGLE ANALYTICS INTEGRATION

\*\*\*\*\*

adapté de <http://blog.janjonas.net/2010-01-31/cakephp-google-analytics-integration>

1) Copier ce code dans src/Template/Element/google-analytics.ctp :

```
<?php
```

```
$gaCode = Configure::read('google-analytics.tracker-code');
```

```
if ($gaCode) {
```



```

$googleAnalytics = <<<EOD
<script type="text/javascript">
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','/www.google-analytics.com/analytics.js','ga');

ga('create', 'UA-45668893-3', 'omp.eu');
ga('send', 'pageview');

</script>
EOD;
echo $googleAnalytics;
}
?>

```

```

/* ANCIEN CODE JAVASCRIPT :
<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src=" + gaJsHost + "google-analytics.com/ga.js" type='text/javascript'%3E%3C/script%3E"));
</script>
<script type="text/javascript">
try {
var pageTracker = _gat._getTracker("$gaCode");
pageTracker._trackPageview();
} catch(err) {}</script>
*/

```

2) Inclure le view element dans src/Template/Layout/default.ctp (juste avant </body>):

```
<?php echo $this->element('google-analytics'); ?>
```

3) Définir le tracker code dans la configuration (/app/Config/core.php):

```
Configure::write('google-analytics.tracker-code', false); // disables Google Analytics
Configure::write('google-analytics.tracker-code', 'YOUR-TRACKING-CODE'); // enables Google Analytics
```

## GENERALITES

\*\*\*\*\*

- URL : `http://localhost/inventirapsvn/cakephp/`
- BD admin : `http://localhost/phpmyadmin`
- LOG : `cakephp/app/tmp/logs/inventirap.log`
- (UPDATE : `cd Inventirap/ ; chmod -R 777 ./cakephp/app/tmp/`)

### - Eclipse config :

Setup syntax highlighting for .ctp files:

Window > Preferences > General > Appearance > content Types > Text > PHP Content type > Add.. , then put in \*.ctp.

### - Cakephp config : `cakephp/app/Config/`

Mode debug ON (pour voir les erreurs et pour vider le cache en cas de changement sur la BD)

`core.php : Configure::write('debug', 1);`

### - Fichier de demarrage :

`cakephp/index.php` (qui pointe sur `cakephp/app/webroot/index.php`)

(Attention, il y a aussi un fichier `cakephp/app/index.php` qui pointe sur `webroot/index.php`)

(ROOT doit pointer sur le dossier `cakephp/`)

Le fichier execute ensuite est `cakephp/lib/Cake/bootstrap.php` qui lance `./Core/App.php`

and so on...

## ACL (Controle d'accès aux ressources)

\*\*\*\*\*

NB: Cette section n'est sans doute plus très à jour ; sur ce sujet, il est préférable de lire le document `docs/userguide/ACL.doc` (ou `.pdf`)

## Synthese sur les droits selon le profil

Profils (roles), dans le sens du pouvoir croissant :

Qq = Utilisateur Quelconque (lambda)

Rp = Responsable

Ad = Administration

Sa = Superadmin

Actions :

C = Create

R = Read (voir, consulter)

U = Update (mettre a jour)

D = Delete (supprimer)

V = Valider un materiel CREATED --> passe alors en statut VALIDATED

A = Demander l'Archivage d'un materiel VALIDATED --> passe alors en statut TOBEARCHIVED

S = Sortir de l'inventaire (Valider une demande d'archivage d'un materiel TOBEARCHIVED) --> passe alors en statut ARCHIVED

E = Exporter

Par default, le superadmin a acces a TOUT

Materiels :

- Qq a les droits C, R (sauf champs admin), U (si createur et sauf champs admin), A, D (si CREATED et owner)

- Rp a les droits C, R (sauf champs admin), U (sauf champs admin), D (si CREATED), V, A, E

- Ad a les droits C, R, U (ssi NOT ARCHIVED), D (si CREATED), V, (A mais inutile car fait directement S sans passer par A), S, E

Suivis et Emprunts :

- Dans tous les cas, on ne doit pas pouvoir emprunter ou suivre un materiel non valide (CREATED)

- Qq a les droits C, R, U (si createur), D (si createur)

- Rp a les droits C, R, U, D

- Ad a les memes droits que Rp

VUES specifiques :

- Acces aux Outils : reserve a Rp et Ad (vue contenant des liens vers differentes ressources telles que utilisateurs, materiels, categories...)

- L'administration a une vue resumee sur la page d'accueil (liens directs vers actions a faire)

- L'administration a une vue enrichie de la liste des materiels :

- filtres (y-compris sur ARCHIVED)
- export en CSV (pour tableur Excel)
- upgrade du statut (validation et sortie)

Autres regles (de gestion) importantes :

- un materiel non valide (CREATED) peut être supprimé uniquement par le créateur de la fiche, le responsable (Roger), et l'administration. Idem pour la mise à jour de cette fiche
- un materiel valide (VALIDATED) n'est pas supprimable
- les champs ADMINISTRATIFS d'un materiel ne sont visibles et modifiables que par l'administration
- par défaut, la liste des matériels affiche tous les matériels SAUF ceux qui sont sortis de l'inventaire (ARCHIVED) qui sont donc masqués. Il est de toutes façons toujours possible (pour l'administration seulement) de les voir, grâce au nouveau filtre "Archives" présent sur la liste des matériels
- la recherche doit s'effectuer dans TOUS les matériels (y-compris ARCHIVED)

## COMMENT CONTOURNER LE LDAP officiel

\*\*\*\*\*

(ATTENTION: CONTENU OBSOLETE A METTRE A JOUR (EP))

(cf Controller/UtilisateursController.php : méthode login())

1) Si on veut tester le LDAP, on peut utiliser le LDAP de la Virtual Machine Upsilon (CentOS 6)

Dans ce cas, il faut ajouter dans la BD les utilisateurs spécifiques suivants :

# Ajout d'utilisateurs de TEST (LDAP upsilon)

```
INSERT INTO utilisateurs (nom, login, email, role, groupes_metier_id) VALUES
('Hillebrand Cedric', 'Cedric', 'Cedric.Hillebrand@irap.omp.eu', 'Super Administrateur', 1),
('Turner Daniel', 'Daniel', 'Daniel.Turner@irap.omp.eu', 'Administration', 1),
('Sky Gin', 'Gin', 'Gin.Sky@irap.omp.eu', 'Responsable', 1),
('Robert Henri', 'Henri', 'Henri.Robert', 'Utilisateur', 1);
```

2) Sinon, le plus simple est d'utiliser un FAUX (fake) LDAP interne à l'application

Dans ce cas, il faut décommenter la variable \$fakeLDAP dans le fichier config.php et ajouter dans la BD les utilisateurs spécifiques suivants :

# Ajout d'utilisateurs de TEST (hors LDAP)

```
INSERT INTO utilisateurs (nom, login, email, role, groupes_metier_id) VALUES
('Hubert SuperAdmin', 'superadmin', 'hubert.superadmin@irap.omp.eu', 'Super Administrateur', 1),
('Pierre Responsable', 'responsable', 'pierre.resp@irap.omp.eu', 'Responsable', 2),
('Jean Administration', 'admin', 'Jean.Administration@irap.omp.eu', 'Administration', 5),
('Jacques Utilisateur', 'user', 'Jacques.Utilisateur@irap.omp.eu', 'Utilisateur', 7);
```

Informations sur le LDAP :

Classe LdapAuthComponent (extends AuthComponent), definie dans : app/Controller/Component/LdapAuthComponent.php

Definition des roles (ACL) : app/Model/Utilisateur.php :

```
private $acceptedRoles = array ('Utilisateur', 'Responsable', 'Administration', 'Super Administrateur');
public function getAuthenticationLevelFromRole($role) {
 if ($role == 'Utilisateur')
 return 1;
 elseif ($role == 'Responsable')
 return 2;
 elseif ($role == 'Administration')
 return 3;
 elseif ($role == 'Super Administrateur')
 return 4;
 return 0;
}
```

Lecture du fichier de config : app/Model/LdapConnection.php :

```
private function checkConfiguration()
```

Workflow du LDAP :

- 1) Page d'accueil : j'entre mon login et pwd
- 2) clic sur bouton "Se Connecter" --> execute l'action Utilisateurs/login (dans app/Controller/UtilisateursController.php)

## COMMENT AJOUTER UNE NOUVELLE TABLE DANS LA BD

\*\*\*\*\*

On explique ici ce qu'il faut faire quand on veut ajouter une nouvelle table dans la base de données, table qui doit être prise en compte dans l'application.

Cette explication est donnée avec 2 exemples.

1) Premier exemple : ajout de la table sur-categorie (EP)

Avant cet ajout, il n'y avait que la table categorie et la table sous-categorie.

Vous trouverez plus de détails sur ce sujet dans la section suivante nommée "COMMENT J'AI FAIT POUR AJOUTER UN 3eme niveau de categorie appele sur\_categorie (ou domaine)"

a) impact sur la BD

- ajout d'une nouvelle table sur\_categories(id,nom) :

```
CREATE TABLE IF NOT EXISTS sur_categories (
 id int(11) NOT NULL AUTO_INCREMENT,
 nom varchar(45) DEFAULT NULL,
 PRIMARY KEY (id),
 UNIQUE KEY nom_UNIQUE (nom)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- ajout dans la table categories d'une cle etrangere sur\_categorie\_id :

```
ALTER TABLE categories
 ADD sur_categorie_id INT(11) NULL DEFAULT NULL,
 ADD CONSTRAINT fk_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO ACTION ON
UPDATE NO ACTION;
```

- ajout dans la table materiels d'une cle etrangere sur\_categorie\_id :

```
ALTER TABLE materiels
 ADD sur_categorie_id INT(11) NOT NULL after designation,
 ADD CONSTRAINT fk_materiels_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO
ACTION ON UPDATE NO ACTION;
```

(attention, il faut d'abord vider les lignes des tables suivis, emprunts, et materiels)

```
delete from suivis;
delete from emprunts;
delete from materiels;
```

Il faut aussi ajouter des sur-categories :

# Ajout de quelques sur-categories

```
insert into sur_categories (id,nom) values
```

```
(1,'Electronique'),
```

```
(2,'Informatique'),
```

```
(3,'Instrumentation')
```

```
(4, 'Logistique'),
```

```
(5, 'Mecanique'),
```

```
(6, 'Optique')
```

```
;
```

# Relier toutes les categories a une sur-categorie (la 1)

```
update categories set sur_categorie_id=1 where id<10;
```

```
update categories set sur_categorie_id=2 where id>=10 and id<20;
```

```
update categories set sur_categorie_id=3 where id>=20;
```

b) impact sur les modeles (cakephp/app/Model)

- Ajout du nouveau modèle SurCategorie.php : copie de Categorie.php avec "hasMany categorie"

- Modif du modèle Categorie.php : + belongsTo="SurCategorie"

c) impact sur les controleurs (cakephp/app/Controller)

- Ajout du nouveau controleur SurCategoriesController.php : minimaliste (comme CategoriesController)

- Modif du controleur CategoriesController.php : +getBySurCategorie()

d) impact sur les vues (cakephp/app/View)

- SurCategorie : no view (scaffold ?)

- Categorie : actuellement no view (scaffold ?)

--> creer une vue (comme SousCategorie)

- + get\_by\_surcategorie.ctp

- + scaffold.form.ctp

e) View/Pages/tools.ctp : ajouter une entree au menu pour les Sur-Categories

f) impact sur TOUTES les vues de Materiel

add/edit/view/index

dans la vue index, remplacer la categorie par la sur-categorie

GROS BOULOT sur la vue ADD/EDIT (+ javascript)

2) Deuxième exemple : ajout de la table type\_suivis (VM)

Un peu plus haut est expliqué comment créer une table, je vais ici expliquer comment créer et remplir tout ce qui est basiquement nécessaire pour l'utiliser, en prenant pour exemple la table type\_suivis.

Après avoir créé la table, on lui crée :

- Un controller : Héritant de ApplicationController, avec une variable \$scaffold qui se chargera de presque tout et une variable \$name avec son nom.
- Un model : Avec la même variable \$name et une variable \$displayField qui correspond a la designation dans la BDD (ex: "nom");  
Il doit contenir la fonction typeSuiviNamelsUnik(\$name) {} (qu'on retrouve dans sites ou organisme) ainsi que le \$validate comprenant les validations necessaires.
- Une vue, selon l'utilité, n'est pas forcément nécessaire grâce au scaffold.

Par la suite, pour lister son contenu via un lien dans "outils" par exemple, il suffit de créer le chemin dans view/pages/tools.ctp.

CREER UN CHAMP DATE (VM)

\*\*\*\*\*

Pour expliquer comment créer un champ date fonctionnel, je vais prendre l'exemple de la Date de reception.

Une fois votre colonne créée dans la table materiel, vous devrez faire des modifications dans :

MaterielController : //Passer la date au format français

```
if(isset($this->request->data['Materiel']['date_reception'])){
 $this->request->data['Materiel']['date_reception'] = date("d-m-Y", strtotime($this->request->data['Materiel']['date_reception'])); }
```



le Model Materiel : Créer un champ de validation adapté à la date, avec un regex. Et surtout remplir le formatage de date à l'américaine :

```
if (isset($this->data['Materiel']['date_reception'])) {
 $originalDate = $this->data['Materiel']['date_reception'];
 $this->data['Materiel']['date_reception'] = date("Y-m-d", strtotime($originalDate));
}
```

La vue Materiel : - Scaffold.view : Repasser la date en français et l'afficher.

- Scaffold.form : Créer le champ date du formulaire

Puis il faut adapter le champ date reception presque partout ou il y a le champ date acquisition.

COMMENT J'AI FAIT POUR AJOUTER UN 3eme niveau de categorie appele sur\_categorie (ou domaine) (EP) ?

\*\*\*\*\*

je me rends compte d'une incoherence de stockage dans la table Materiels.

En effet, quand on saisit un materiel, on doit choisir une categorie ET une sous-categorie.

Du coup, les 2 id (categorie + sous-categorie) sont stockes dans la table Materiels...

Or, c'est inutile car la sous-categorie suffit a determiner la categorie...

Cette redondance pourrait meme amener des incoherences dans la table Materiels si par exemple on fait des modifications dans les tables categories et sous\_categories sans les repercuter dans la table materiels !!

Je suppose que c'est un choix de facilite qui a ete fait par upsilon.

Donc, si vous etes ok, et a moins que Upsilon me dise qu'il y avait une bonne raison de faire ce choix (j'en doute), je propose de modifier le code pour que seule la sous-categorie soit stockee (on ne stocke plus la categorie).

En plus, cela simplifiera l'ajout du 3eme niveau "sur-categorie" puisque lui-meme n'aura pas besoin d'etre stocke etant donne qu'il est automatiquement determine par la categorie.

On aura alors :

sous\_categorie\_id -> categorie\_id -> sur\_categorie\_id

Avec seulement la sous\_categorie, on pourra determiner la categorie, et donc la sur\_categorie.

Du coup, si je fais cette modif, on pourrait meme se permettre de demarrer officiellement INVENTIRAP rapidement.  
En effet, l'ajout de la sur-categorie ne change rien au contenu des tables "administratives" (materiels, emprunts, suivis),  
et donc on pourra le faire plus tard, meme apres que l'administration ait commence a remplir la BD.  
Ca sera totalement transparent.

1) suppression de la redondance (categorie\_id) dans la table materiels

a) impact sur la vue index de Materiel  
add/edit/view/index

2) ajout d'une sur-categorie

a) impact sur la BD

- ajout d'une nouvelle table sur\_categories(id,nom) :

```
CREATE TABLE IF NOT EXISTS sur_categories (
 id int(11) NOT NULL AUTO_INCREMENT,
 nom varchar(45) DEFAULT NULL,
 PRIMARY KEY (id),
 UNIQUE KEY nom_UNIQUE (nom)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- ajout dans la table categories d'une cle etrangere sur\_categorie\_id :

```
ALTER TABLE categories
 ADD sur_categorie_id INT(11) NULL DEFAULT NULL,
 ADD CONSTRAINT fk_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO ACTION ON
UPDATE NO ACTION;
```

- ajout dans la table materiels d'une cle etrangere sur\_categorie\_id :

```
ALTER TABLE materiels
 ADD sur_categorie_id INT(11) NOT NULL after designation,
 ADD CONSTRAINT fk_materiels_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO
ACTION ON UPDATE NO ACTION;
```

(attention, il faut d'abord vider les lignes des tables suivis, emprunts, et materiels)

```
delete from suivis;
delete from emprunts;
delete from materiels;
```

Il faut aussi ajouter des sur-categories :

# Ajout de quelques sur-categories

```
insert into sur_categories (id,nom) values
```

```
(1,'Electronique'),
```

```
(2,'Informatique'),
```

```
(3,'Instrumentation')
```

```
(4, 'Logistique'),
```

```
(5, 'Mecanique'),
```

```
(6, 'Optique')
```

```
;
```

# Relier toutes les categories a une sur-categorie (la 1)

```
update categories set sur_categorie_id=1 where id<10;
```

```
update categories set sur_categorie_id=2 where id>=10 and id<20;
```

```
update categories set sur_categorie_id=3 where id>=20;
```

b) impact sur les modeles (cakephp/app/Model)

- SurCategorie.php : copie de Categorie.php avec "hasMany categorie"

- Categorie.php : + belongsTo="SurCategorie"

c) impact sur les controleurs (cakephp/app/Controller)

- SurCategoriesController.php : minimaliste (comme CategoriesController)

- CategoriesController.php : +getBySurCategorie()

d) impact sur les vues (cakephp/app/View)

- SurCategorie : no view (scaffold ?)

- Categorie : actuellement no view (scaffold ?)

--> creer une vue (comme SousCategorie)

- + get\_by\_surcategorie.ctp

- + scaffold.form.ctp

e) View/Pages/tools.ctp : ajouter une entree au menu pour les Sur-Categories

f) impact sur TOUTES les vues de Materiel

add/edit/view/index

dans la vue index, remplacer la categorie par la sur-categorie

GROS BOULOT sur la vue ADD/EDIT (+ javascript)

## TESTS

\*\*\*\*\*

### A - EXECUTION

\*\*\*\*\*

Prérequis : PHPUNIT 3 doit être déjà installé et accessible depuis la console (phpunit -version) via le fichier php.ini  
(ATTENTION : cakephp 2.x n'est pas compatible avec PHPUnit 4)

Avec XAMPP (conseillé) : PHPUnit 3 est déjà inclus

Avec MAMP (+ difficile) : pour installer PHPUnit, suivre cette documentation :

<http://www.dolinaj.net/software-installation/mac/how-to-install-phpunit-with-mamp-on-mac/>

Procédure à suivre pour pouvoir exécuter les tests unitaires et fonctionnels livrés avec le logiciel :

1) Ajouter une nouvelle configuration de base de données dans votre fichier app/Config/database.php pour la BD de test

Exemple de configuration :

```
public $test = array(
 'datasource' => 'Database/Mysql',
 'persistent' => false,
 'host' => 'localhost',
 'database' => 'test_labinvent',
```

```
'login' => 'root',
'password' => "",
);
```

## 2) Créer la BD de test (vide)

A l'aide de phpmyadmin ou bien avec un client mysql quelconque, créer une BD de test vide que vous nommerez avec le même nom que celui donné dans la configuration ci-dessus, soit pour l'exemple, "test\_labinvent"  
Syntaxe SQL : "create database test\_labinvent"

## 3) Passer en mode "debug"

Dans votre fichier de configuration labinvent.php, mettez votre paramètre "debug" à 1 ou 2 (mais pas à 0) :

```
'debug' => 2,
```

## 4) Se connecter à l'application

Les test ne passeront pas si vous n'êtes pas connectés

## 5) Exécuter les tests

### a) Exécution depuis l'application (conseillé) :

ATTENTION : vous devez être logué pour que les tests passent !!!

Il suffit d'aller à l'URL /test.php de votre installation du logiciel LabInvent

(vous pouvez aussi essayer l'URL "/app/webroot/test.php", ou encore "/cakephp/test.php")

Cette page de tests se trouve dans cakephp/app/webroot/

Vous avez alors accès à 2 types de tests :

- App : Tests ==> les tests écrits pour tester l'application Labinvent

- Core : Tests ==> les tests fournis avec cakephp pour tester le framework

Pour exécuter tous les tests liés à l'application Labinvent (à faire systématiquement avant de commiter tout changement) :

cliquer sur "Tests" sous "App", puis sur "AllTests"

("AllController" exécute tous les tests de controleurs ; "AllModel" exécute tous les tests de modèles)

### b) Execution depuis la console :

Aller dans le repertoire app/

Pour tester le controleur MaterielsController :

./Console/cake test app Controller/MaterielsController

## B - ECRITURE DE NOUVEAUX TESTS

\*\*\*\*\*

cf <http://book.cakephp.org/2.0/en/development/testing.html>

Aller dans app/Test/

Ce dossier contient l'arborescence suivante :

- Case/ : les tests
  - Controller/ : les tests de controleurs
  - Model/ : les tests de modèles
  - View/ (peu ou pas utilisé) : les tests de vues (ou de helpers)
  - AllControllerTest.php : exécution de tous les tests de controleurs
  - AllModelTest.php : exécution de tous les tests de modèles
  - AllTestsTest.php : exécution de TOUS les tests
  
- Fixture/ : les différentes initialisations nécessaires dans la BD de test pour pouvoir executer les tests  
Ces "fixtures" sont automatiquement executees AU DEBUT de chaque test.  
Ce dossier contient un fichier pour chaque table pour laquelle on a besoin d'une "fixture".

### 1) Les "fixtures"

La façon la plus basique de creer une fixture pour une table donnee est de la realiser automatiquement a partir d'une copie de la table de la vraie BD. Par exemple, pour que la table "categories" de la BD de test contienne la meme chose que la table de la vraie BD, il suffit de creer un fichier `CategorieFixture.php` contenant ceci :

```
class CategorieFixture extends CakeTestFixture {
 public $import = array('model' => 'Categorie', 'records' => true);
}
```

Au demarrage des tests, cette table sera chargee automatiquement avec les vraies donnees.

A la fin des tests, cette table sera vidée.

Dans le cas particulier de la table "materiels", on prefere l'initialiser nous-memes avec des valeurs choisies.  
Exemple d'une fixture avec 2 materiels (dans le fichier MaterielFixture.php) :

```
class MaterielFixture extends CakeTestFixture {

 public $import = 'Materiel'; // import only structure, no record

 public $records = array(
 array(
 'designation' => 'matos1',
 'sur_categorie_id' => 1,
 'categorie_id' => 11,
 'materiel_administratif' => 0,
 'materiel_technique' => 1,
 'status' => 'CREATED',
 'nom_createur' => 'Pallier Etienne',
 'nom_modificateur' => 'Jean Administration',
 'nom_responsable' => 'Jacques Utilisateur',
 'email_responsable' => 'Jacques.Utilisateur@irap.omp.eu',
),
 array(
 'designation' => 'matos2',
 'sur_categorie_id' => 1,
 'categorie_id' => 11,
 'materiel_administratif' => 0,
 'materiel_technique' => 1,
 'status' => 'CREATED',
 'nom_createur' => 'Pallier Etienne',
 'nom_modificateur' => 'Jean Administration',
 'nom_responsable' => 'Jacques Utilisateur',
 'email_responsable' => 'Jacques.Utilisateur@irap.omp.eu',
),
);
};
```

```
}
```

## 2) Les tests

Prenons l'exemple des tests écrits pour le contrôleur des matériels (MaterielsController). Il devra s'appeler MaterielsControllerTest et aura la structure suivante :

```
class MaterielsControllerTest extends ControllerTestCase {

 // Liste des fixtures à charger avant l'exécution des tests
 public $fixtures = array('app.materiel', 'app.sur_categorie', 'app.categorie', 'app.sous_categorie',
 'app.groupes_thematique', 'app.groupes_metier', 'app.suivi', 'app.emprunt'
);

 // Initialisations diverses à faire avant chaque test
 public function setUp() {
 parent::setUp();
 }

 // un 1er test
 public function testMonPremier() {
 $result = $this->testAction(...);
 $this->assert...('resultat attendu', $result);
 }

 // un 2eme test
 public function testMonDeuxieme() {
 $result = $this->testAction(...);
 $this->assert...('resultat attendu', $result);
 }

 ...
}
```



Voir le vrai fichier Test/Case/Controller/MaterielsControllerTest.php

### 3) Execution

Exemple avec l'execution du test MaterielsControllerTest

a) Execution depuis le site web :

/test.php?case=Controller%2FMaterielsController

Ajouter &debug=1 à l'url pour voir tous les messages de debug

b) Execution depuis la console :

Dans le repertoire app : ./Console/cake test app Controller/MaterielsController

Ajouter --debug pour voir tous les messages de debug

## DATE PICKERS

\*\*\*\*\*

Pour fonctionner, le datePicker fait appel dans la page "View/Layout/default" à 3 scripts (jquery-1.5.2.js, jquery-1.8.12.js, DatepickerConfig.js) présents dans le repertoire "webroot/js/" et à un fichier "Theme" (smoothness.css) présent dans le repertoire "webroot/css/"

Le thème global peut très facilement être changé en téléchargeant le fichier css de son choix, à cette adresse: "<http://jqueryui.com/themeroller/>" et en remplaçant celui se trouvant dans "webroot/css/"

Les options du datePicker sont modifiables dans le fichier "webroot/js/DatepickerConfig.js" et sont assez explicites. Malgré cela, pour plus de précisions, la doc est facilement consultable à cette adresse: "<http://jqueryui.com/datepicker/>"

# 11. Cycle de développement à respecter

(11 commandements)

Je veux apporter un changement (correction ou evolution) au projet, comment dois-je faire ?

1) Incrementer la version du projet, a la fin du fichier cakephp/app/View/Layouts/default.ctp, et mettre à jour la date

2) Selectionner le changement a apporter (une demande) dans le Redmine du projet (<https://projects.irap.omp.eu/projects/inventirap>) :

- soit depuis la liste des demandes : onglet Demandes

- soit depuis la roadmap : onglet Roadmap, cocher la case "Anomalie", cliquer sur Appliquer, puis "version 1.3" (la version en cours depuis fin 2012)

Commencer de préférence par les "anomalies" parmi celles qui ont la plus haute priorité (et faire les "évolutions" dans un 2ème temps).

Choisir une demande et cliquer dessus pour aller sur sa fiche detaillee et voir le travail a faire.

Cliquer sur "mettre a jour", selectionner le statut "En cours", modifier eventuellement d'autres champs..., puis cliquer sur "Soumettre".

Noter l'URL de cette fiche (par exemple : <https://projects.irap.omp.eu/issues/1050>)

NB : Si la demande n'existe pas encore, la creer :

- cliquer sur l'onglet "Nouvelle demande"

- Selectionner "Anomalie" ou "Evolution"

- Positionner le statut a "En cours" si on veut travailler aussitot dessus (sinon, laisser a "Nouveau")

- Completer le reste de la fiche de demande (mettre "Assigne a" a "<<moi>>", choisir la version cible "version 1.3" ou "version 1.4", ...)

- cliquer sur le bouton "Creer"

3) Verifier que cette nouvelle demande apparait bien dans la roadmap (cliquer sur onglet "Roadmap", ..., puis version 1.3)

ÉTAPE OPTIONNELLE MAIS FORTEMENT CONSEILLÉE :

4) Ecrire un test qui vérifie que cette fonctionnalité ne marche pas encore (bug) ou bien n'est pas encore implémentée

On utilise ici l'approche TDD (Test Driven Development)

Ce test ne devrait pas passer (il est au rouge) ; il passera plus tard, quand on aura écrit le code nécessaire.

Bien sur, c'est dans la mesure du possible, car on ne peut pas TOUT tester.

Dans tous les cas, il faut écrire un test qui s'approche le plus possible de la réalité à tester.

Voir pour cela la section "TESTS" (à la fin de ce document)

5) Faire les changements necessaires dans le code Php

Si ce changement implique aussi un changement dans la base de donnees,

copier le script SQL correspondant à ce changement dans un fichier database/update/db-update-YYYY-MM-DD.sql

portant la date du jour (voir des exemples dans database/update/old/).

Plus tard, il faudra penser à intégrer ce changement dans le script general de creation de la BDD database/BDD\_IRAP.sql

(et/ou éventuellement les fichiers Insert\_ TablesFunct.sql, Insert\_ Users.sql, et Upd\_ TableConstraints.sql)

et donc deplacer le script de modification database/update/db-update-YYYY-MM-DD.sql dans database/update/old/db-update-YYYY-MM-DD.sql

6) Tester manuellement ce changement jusqu'a ce qu'il soit totalement OK

a) faire quelques tests manuels

b) Le test écrit à l'étape (4) doit maintenant passer (il est au vert).

Il doit être inclus dans l'ensemble des tests accessibles par le lien "AllTests".

c) Test de non régression : afin de s'assurer que cette modification du code n'entraîne aucune régression sur le reste du code, tous les autres tests écrits avant doivent aussi passer (vert) : pour cela, exécuter l'url /test.php?case=AllTests

7) Compléter le fichier /README.txt, section HISTORIQUE DES VERSIONS (fichier situé à la racine du projet)

Y mettre le même commentaire que ce que tu mettras lors du commit

8) Mettre a jour TON code (en mode console, "svn update" depuis la racine du projet, ou bien depuis Eclipse, clic droit sur le projet, "Team/Update")

C'est important, au cas ou quelqu'un d'autre aurait fait des modifs (avant ou en meme temps que toi), et pour etre bien sur d'avoir la derniere version

9) Faire un "commit" du code, en collant dans le commentaire l'URL de la demande realisee (exemple : <https://projects.irap.omp.eu/issues/1050>)

10) Fermer la demande sur redmine

Sur la fiche detaillee, cliquer sur "Mettre a jour", changer le statut a "ferme", changer "% realise" a 100%, (copier l'URL de la fiche), cliquer sur Soumettre

11) Si c'est un changement important, le répercuter sur le site officiel

Le changement est important si c'est une anomalie ou bien si c'est une évolution attendue.

Demander alors au service informatique de le répercuter sur l'installation officielle (faire un "svn update", mail à [loic.jahan@irap.omp.eu](mailto:loic.jahan@irap.omp.eu)).

Si ce changement implique aussi la base de données, donner la directive que le fichier database/update/db-update-YYYY-MM-DD.sql doit être exécuté sur leur BDD.

Si ce changement impliquer une modification du fichier de configuration labinvent.php, donner la procédure à suivre dans le mail.

## 12. Documentation Technique (pour les devs)

Cette doc est en cours de migration à l'intérieur du présent document.

En attendant, voici les liens vers la doc d'origine :

- [Doc technique](#)
- [Doc de développement](#)

# 12.1. Modèle de données (Data model)

Schéma de la base de données (v2.6) :

