Module M2105 - RT web dyna - TD/P 4

Suivre Procédures pour install de Xampp, composer, Ubiquity pour mettre en place Xampp et Ubiquity

Notions abordées

- Gestion des droits
- Connexion
- Mise à jour de données

Reprise du projet existant

Pour intégrer les dernières mises à jour du framework et des librairies utilisées :

En invite de commande, depuis le dossier du projet faire :

```
composer update
```

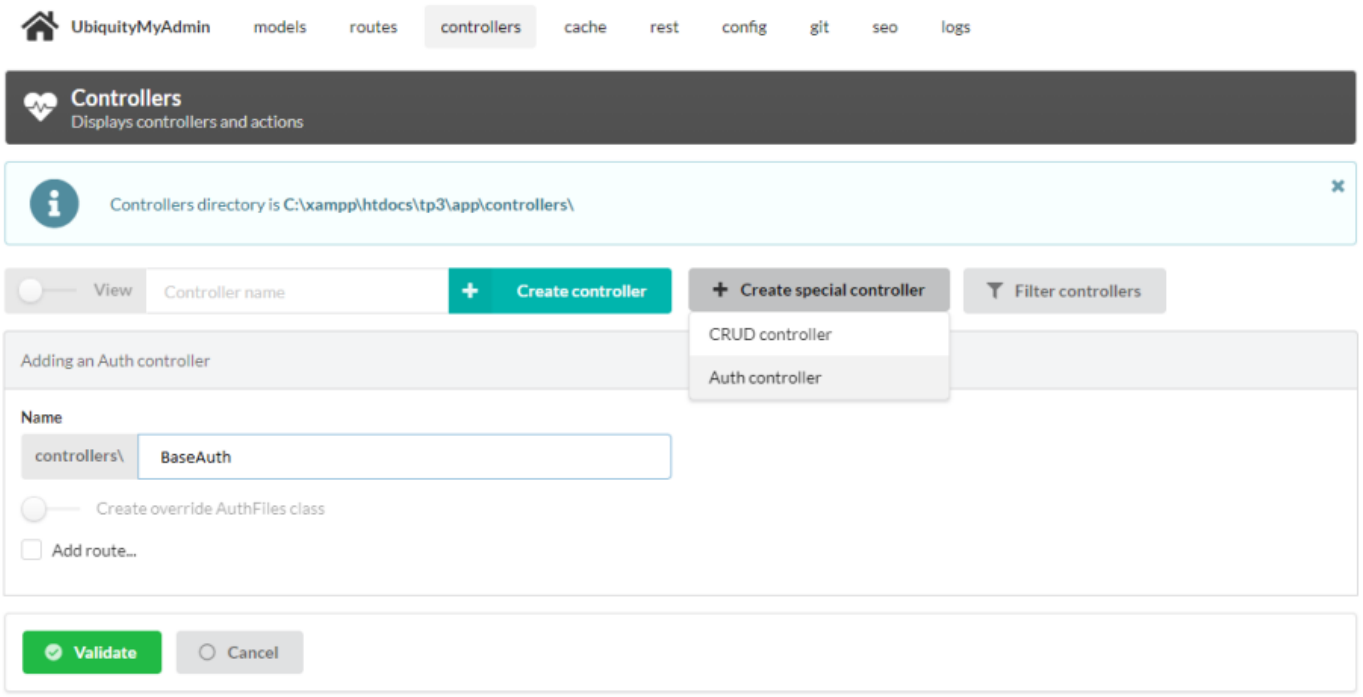
Authentification

L'authentification permet :

- La connexion d'utilisateurs, dont le profil sera mémorisé en session
- L'accès contrôlé aux ressources (contrôleur)
- La déconnexion et le contrôle de fin de session

-- Création d'un contrôleur pour l'authentification

A partir de l'interface d'administration, partie **controllers**, choisir **Create special controller/Auth controller** :



-- Implémentation du comportement

La connexion

L'utilisateur se connecte avec le couple email/passord, validé par un post de formulaire, et dont l'existence est vérifiée dans la base de données :

```
class BaseAuth extends \Ubiquity\controllers\auth\AuthController{

...
    protected function _connect() {
        if(URequest::isPost()){
            $email=URequest::post($this->_getLoginInputName());
            $password=URequest::post($this->_getPasswordInputName());
            $user=DAO::getOne("models\\User", "email='{$email}'");
            if(isset($user) && $user->getPassword()==$password){
                return $user;
            }
        }
        return;
    }
...
}
```

Action post connection

Si la connexion a réussi, l'instance d'utilisateur retournée est stockée en session, puis l'utilisateur connecté est redirigé vers la page sollicitée initialement, ou vers une page par défaut :

```
class BaseAuth extends \Ubiquity\controllers\auth\AuthController{
```

```
...
    protected function onConnect($connected) {
        $urlParts=$this->getOriginalURL();
        USession::set($this->_getUserSessionKey(), $connected);
        if(isset($urlParts)){
            Startup::forward(implode("/",$urlParts));
        }else{
$this->forward("controllers\\Organizations","display",[$connected->getOrganization(
)->getId()],true,true);
        }
    }
...
}
```

Vérification connexion

La méthode de vérification de la validité de la connexion détermine si un utilisateur est correctement connecté (mémorisé en session par exemple)

```
class BaseAuth extends \Ubiquity\controllers\auth\AuthController{

...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::isValidUser()
     */
    public function _isValidUser() {
        return USession::exists($this->_getUserSessionKey());
    }
...
}
```

-- Utilisation de l'authentification

Pour utiliser le contrôleur d'authentification créé sur le contrôleur **Organizations**, ajouter le trait **WithAuthTrait** dans la classe **Organizations**

```
...
use Ubiquity\controllers\auth\WithAuthTrait;
...
class Organizations extends ControllerBase{
    use WithAuthTrait;
...
}
```

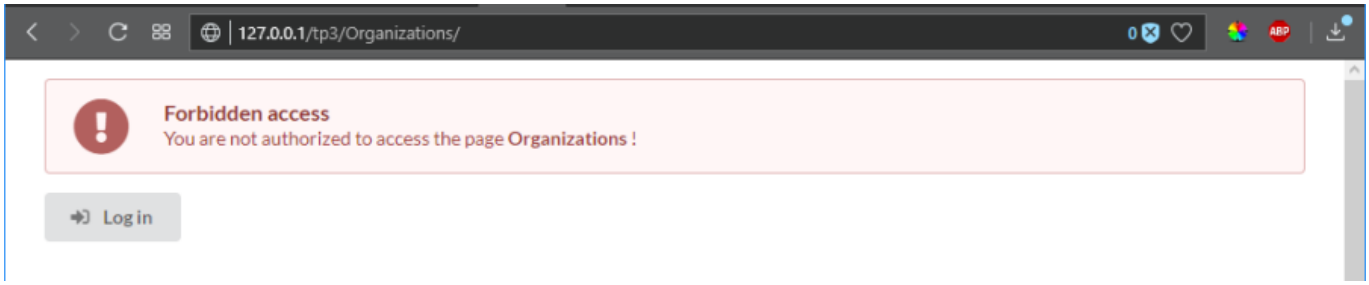
Implémenter ensuite la méthode **getAuthController** :

```
...
```

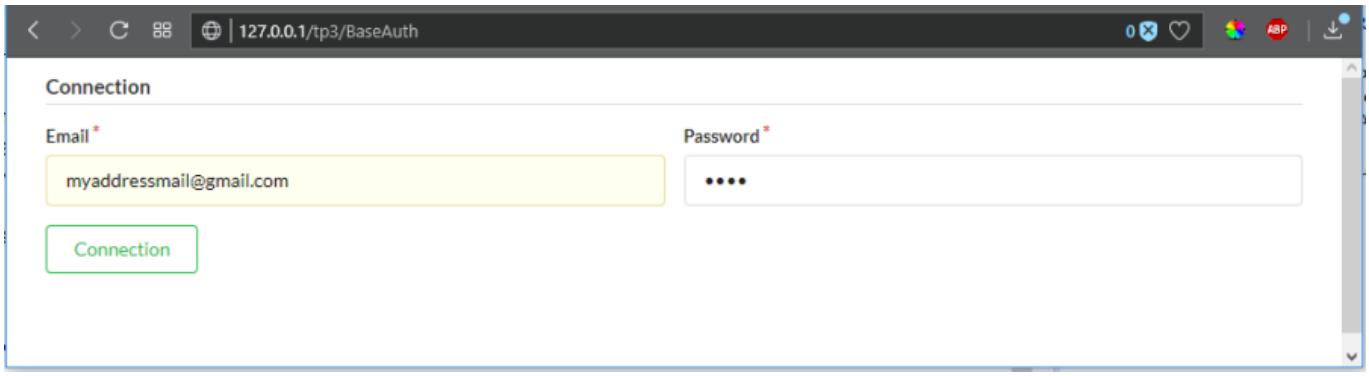
```
use Ubiquity\controllers\auth\AuthController;
use Ubiquity\controllers\auth\WithAuthTrait;
...
class Organizations extends ControllerBase{
    use WithAuthTrait;
...
    protected function getAuthController(): AuthController {
        return new BaseAuth();
    }
...
}
```

L'authentification par défaut est maintenant opérationnelle sur la classe **Organizations** :

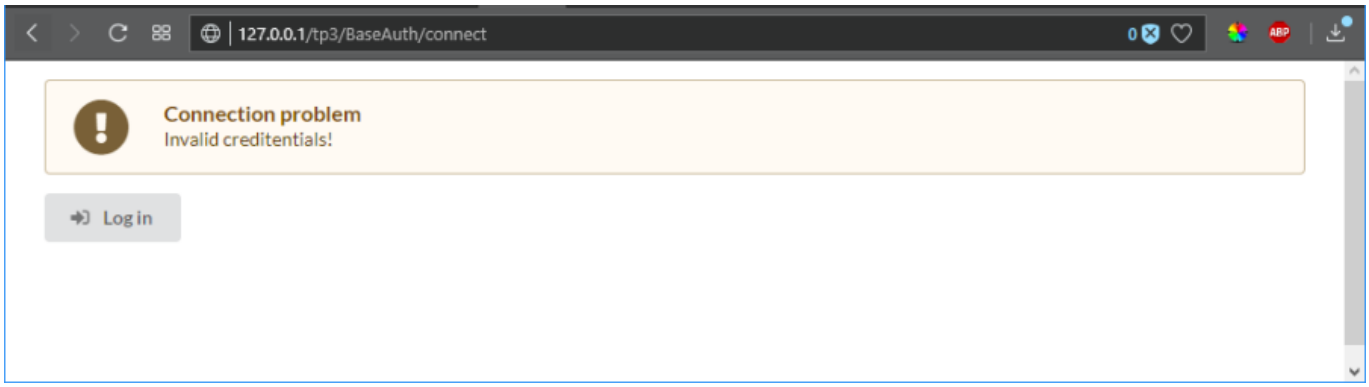
Accéder à l'adresse **/organizations** ou à toute autre action à partir du contrôleur **organizations** :



Cliquer sur **Log in** :



Si la connexion est refusée, le message suivant s'affiche :



-- Affichage de l'utilisateur connecté

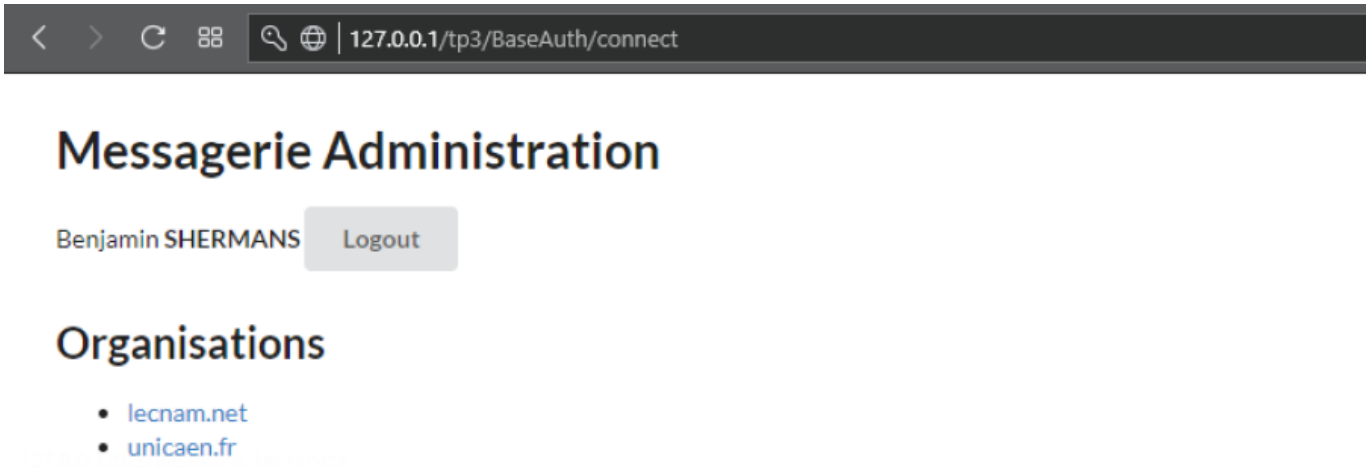
Modifier le template de base pour y intégrer la variable **_userInfo** :

```
{% block header %}
<style>
    a.active{
        font-weight: bold;
        color: red;
    }
    a.active::after {
        content: " →";
    }
</style>
<h1>Messagerie Administration</h1>
{% endblock %}
{% block auth %}
    {{ _userInfo | raw }}
{% endblock %}
{% block message %}
    {{ message | raw }}
{% endblock %}
{% block body %}
{% endblock %}
{{ script_foot | raw }}
```

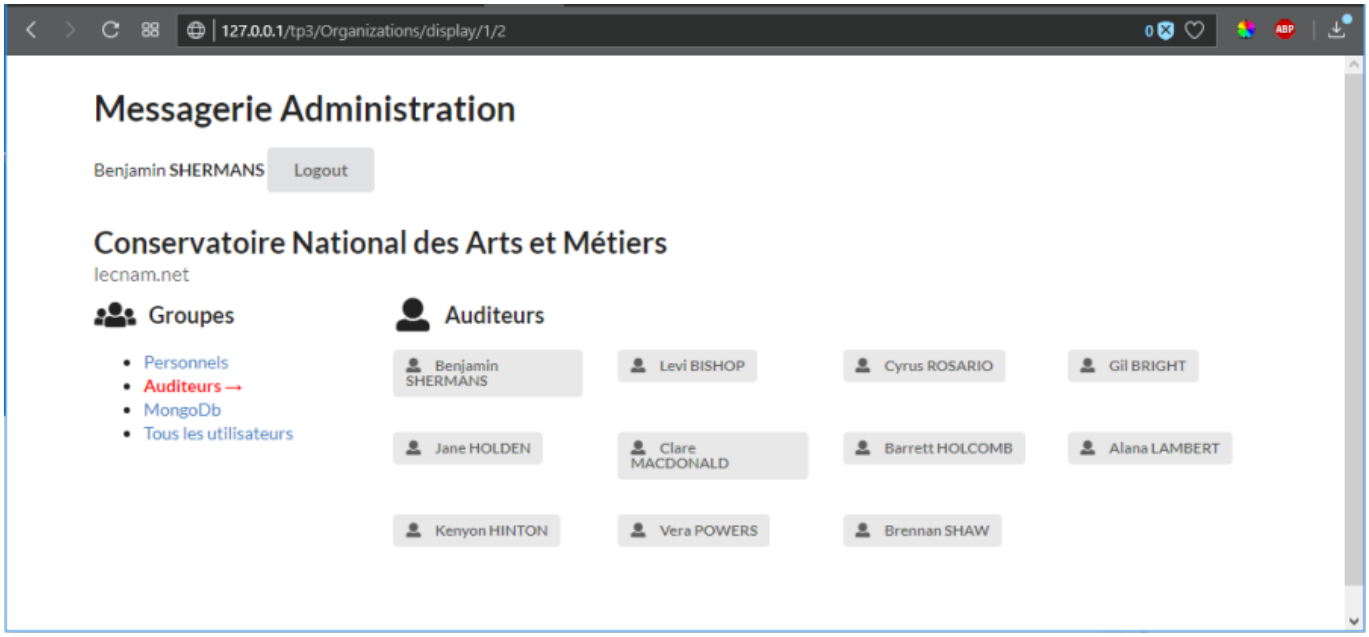
En cas de connexion avec des identifiants corrects :

- email : **benjamin.sherman**
- password : **OWC09RSW6AE**

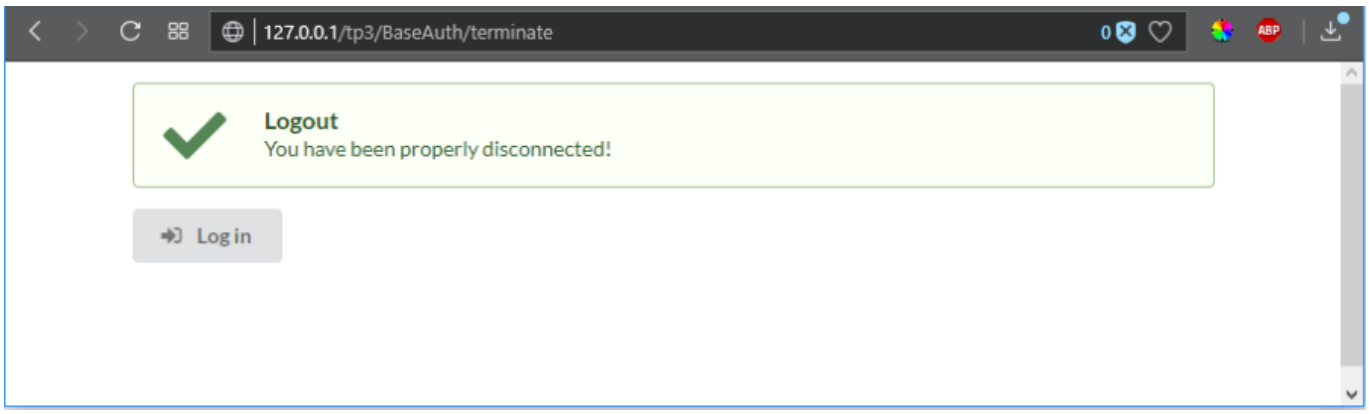
La page **/organizations** affiche :



Et les autres actions du contrôleur affichent également l'authentification :

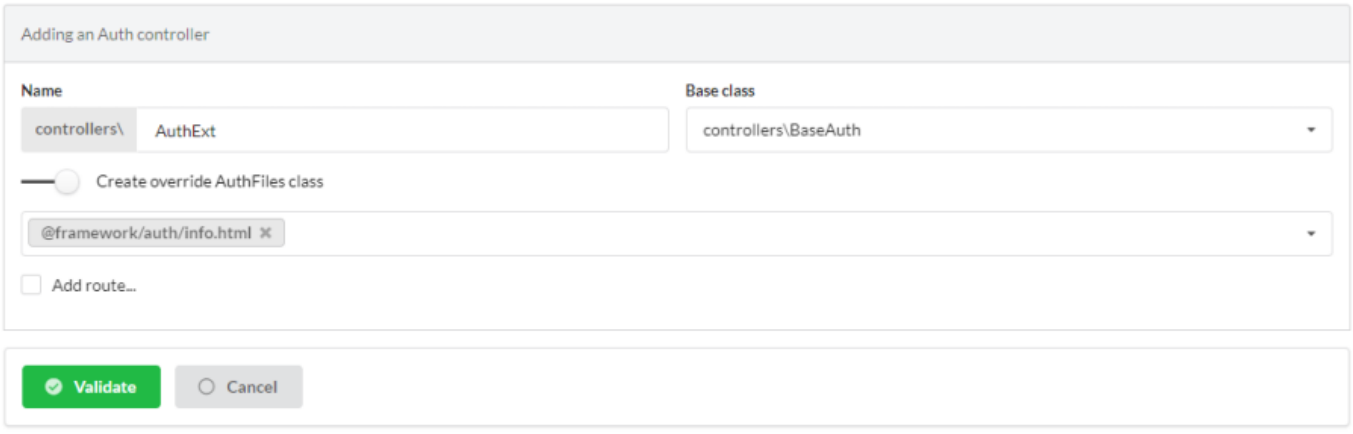


Cliquer sur **Logout** :



-- Personnalisation de l'affichage

A partir de l'interface d'administration, créer un nouveau controller de type **AuthController** héritant du contrôleur **BaseAuth** :



Association de AuthExt

Associer le controller **AuthExt** au contrôleur **Organizations** :

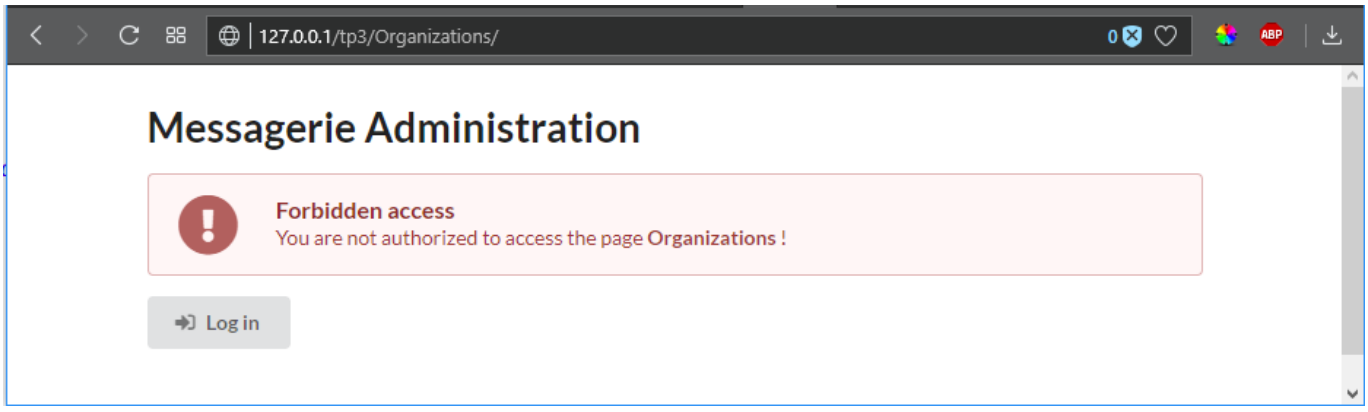
```
...
    protected function getAuthController(): AuthController {
        return new AuthExt();
    }
...
```

Template de base

Modifier le fichier **AuthExtFiles** pour définir le template de base utilisé par les templates de **AuthExt** :

```
<?php
namespace controllers\auth\files;

use Ubiquity\controllers\auth\AuthFiles;
 /**
 * Class AuthExtFiles
 **/
class AuthExtFiles extends AuthFiles{
    public function getViewInfo(){
        return "AuthExt/info.html";
    }
    public function getBaseTemplate(){
        return "base.html";
    }
}
```

Les pages intègrent maintenant le template de base :



Personnalisation des templates

Modifier les blocks **_before**, **_after** et **_logoutCaption** du template **views/AuthExt/info.html** affichant l'information liée à l'utilisateur connecté :

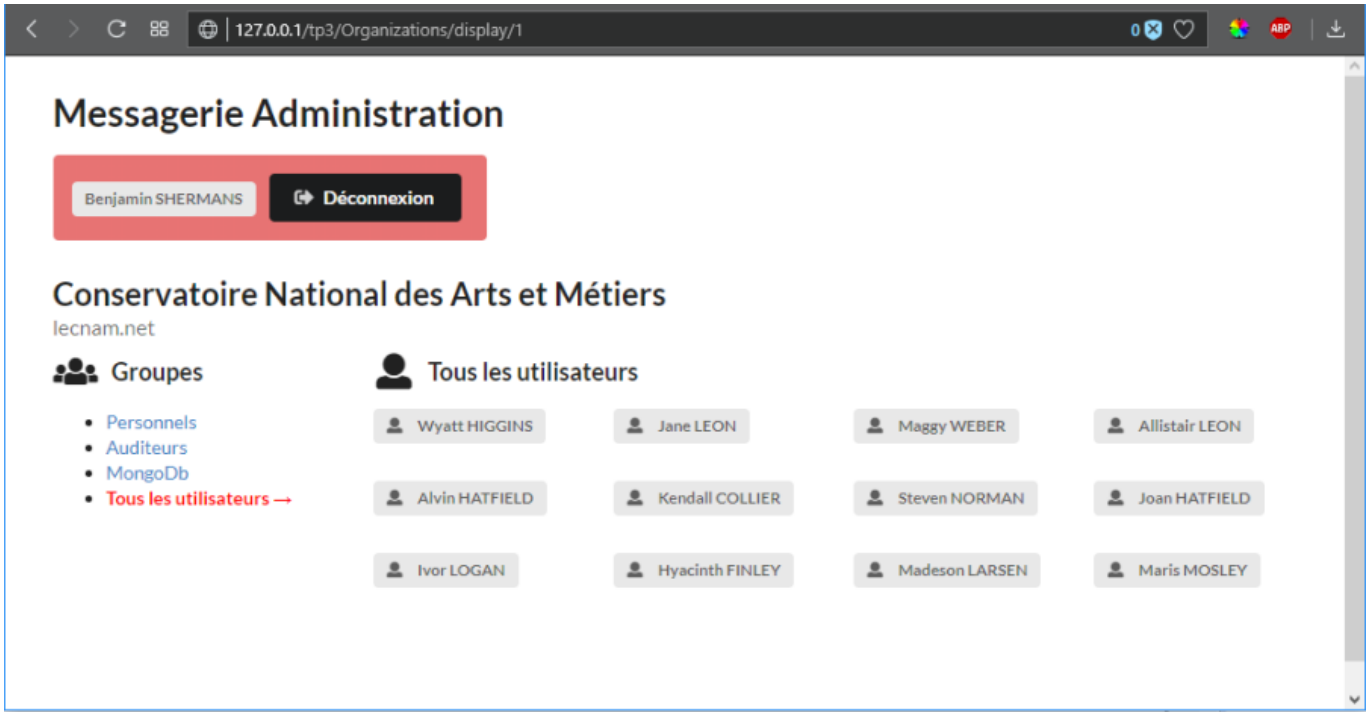
```
{% extends "@framework/auth/info.html" %}
{% block _before %}
<div class="ui tertiary inverted red compact segment">
{% endblock %}
{% block _userInfo %}
    {{ parent() }}
{% endblock %}
{% block _logoutButton %}
    {{ parent() }}
{% endblock %}
{% block _logoutCaption %}
    <i class="ui sign out icon"></i>Déconnexion
{% endblock %}
{% block _loginButton %}
    {{ parent() }}
{% endblock %}
{% block _loginCaption %}
    {{ parent() }}
{% endblock %}
{% block _after %}
</div>
{% endblock %}
```

Placement de la zone _infoUser

Surdéfinir la méthode **_displayInfoAsString** de la classe **AuthExt** :

```
class AuthExt extends \controllers\BaseAuth{

    public function _getBaseRoute() {
        return 'AuthExt';
    }
    protected function getFiles(): AuthFiles{
        return new AuthExtFiles();
    }
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::_displayInfoAsString()
     */
    public function _displayInfoAsString() {
        return true;
    }
}
```

Après connexion, la zone info utilisateur apparaît maintenant localisée à l'endroit où elle a été placée dans le template :



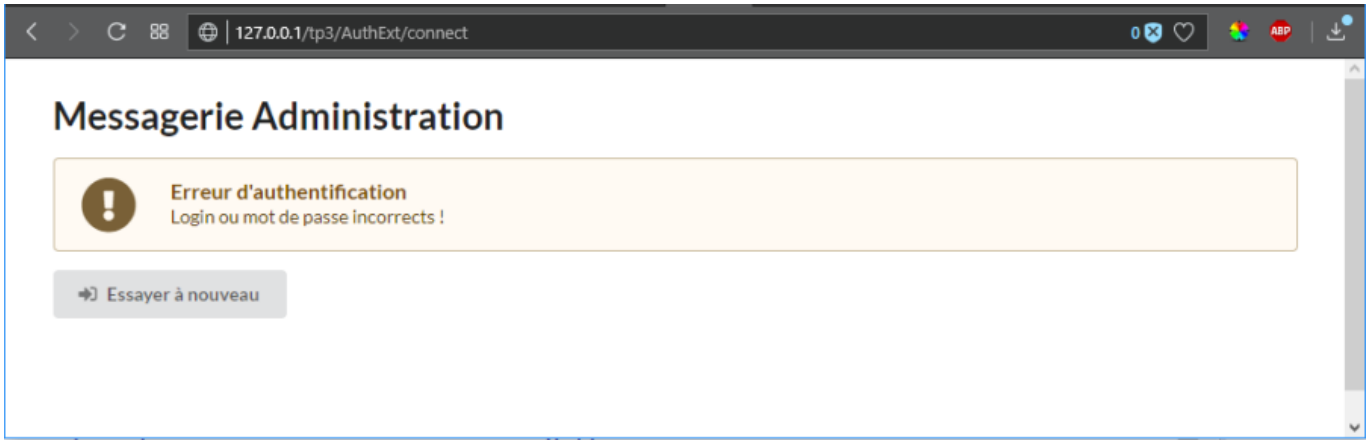
-- Personnalisation des messages

A partir d'Eclipse, dans la classe **AuthExt**, choisir le menu **source/override-implement methods...** :

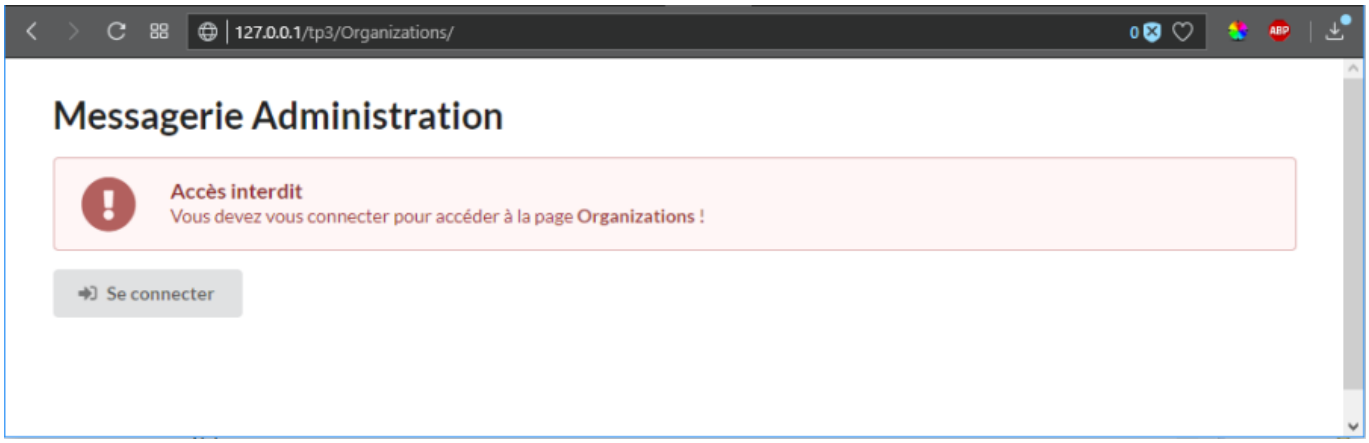
Sélectionner la méthode **badLoginMessage** permettant de modifier le message apparaissant sur une erreur de connexion, et modifier le code comme suit :

```
class AuthExt extends \controllers\BaseAuth{

...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::badLoginMessage()
     */
    protected function badLoginMessage(\Ubiquity\utils\flash\FlashMessage
$fMessage) {
        $fMessage->setTitle("Erreur d'authentification");
        $fMessage->setContent("Login ou mot de passe incorrects !");
        $this->_setLoginCaption("Essayer à nouveau");
    }
...
}
```

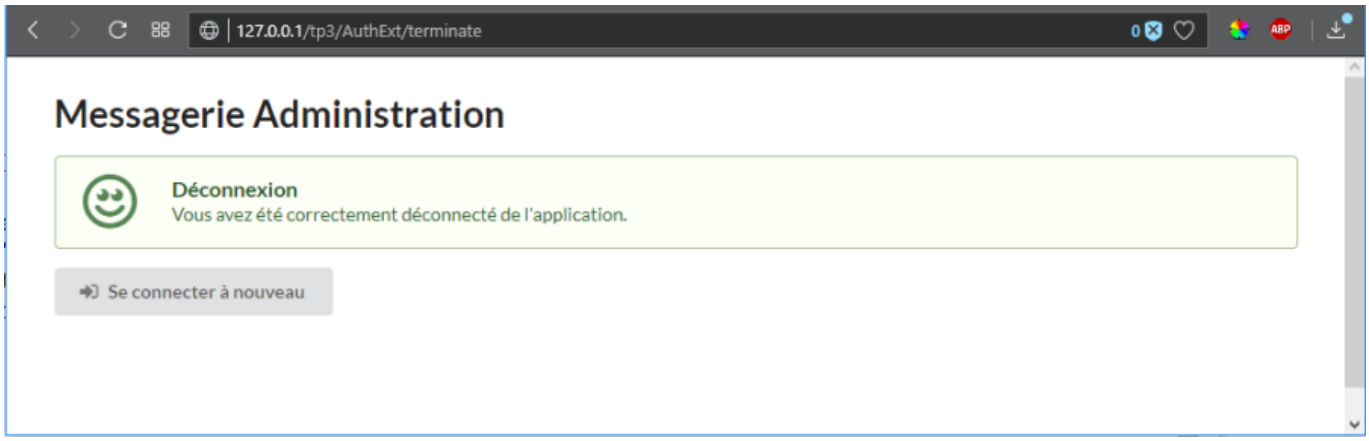
Tester le résultat en entrant un couple email/password non valid :



Surdéfinir la méthode **noAccessMessage** pour obtenir le résultat :
Vous pourrez utiliser la variable **{url}** pour faire référence à l'url à laquelle l'utilisateur tente d'accéder.



Surdéfinir la méthode **terminateMessage** pour obtenir le résultat suivant :



-- Personnalisation du contrôle

On souhaite affiner le contrôle d'accès des utilisateurs, et faire en sorte qu'un utilisateur ne puisse accéder qu'à l'organisation à laquelle il appartient. Surdéfinir la méthode **_isValidUser** de la classe **AuthExt** :

```
class AuthExt extends \controllers\BaseAuth{
...
    public function _isValidUser() {
```

```
        $valid=parent::_isValidUser ();
        if($valid && $this->_action=="display" &&
$this->_controller=="controllers\Organizations"){
            $user=USession::get($this->_getUserSessionKey());
            $valid=$user->getOrganization()->getId()==$this->_actionParams[0];
            if(!$valid){
                $this->_setNoAccessMsg("Vous n'appartenez pas à cette
organisation","Accès non autorisé");
                $this->_setLoginCaption("Se connecter avec un autre compte");
                $bt=$this->jquery->semantic()->htmlButton("btReturn","Retourner à
la page précédente");
                $bt->onClick("window.history.go(-1); return false;");
            }
        }
        return $valid;
    }
...
}
```

Effectuer un commit+push vers gitHub

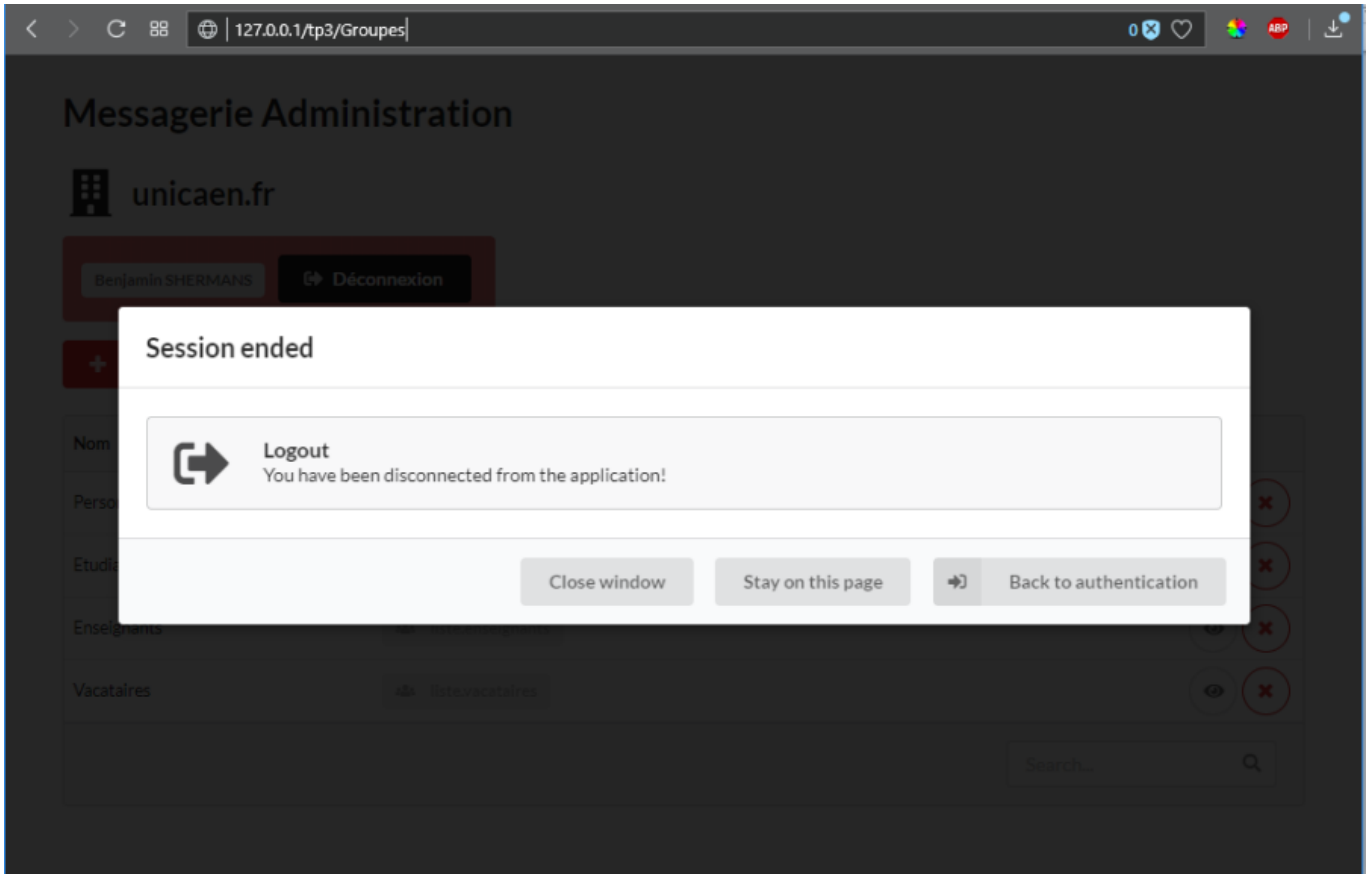
-- Auto-vérification connexion

Surdéfinir la méthode **_checkConnectionTimeout** de la classe **AuthExt** : cette méthode déconnecte toutes les pages du site ouvertes sur le navigateur si la session a été fermée. Dans le cas présent, elle lance un script de vérification de la connexion toutes les 10 secondes.

```
class AuthExt extends \controllers\BaseAuth{
...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::_checkConnectionTimeout()
     */
    public function _checkConnectionTimeout() {
        return 10000;
    }
...
}
```

Pour Tester :

- Se connecter à l'application dans un onglet.
- Accéder à l'application dans un autre onglet du navigateur.
- Se déconnecter de l'application dans cet onglet
- Retourner au premier onglet, et attendre l'apparition du message suivant :



-- Limitation des tentatives de connexion

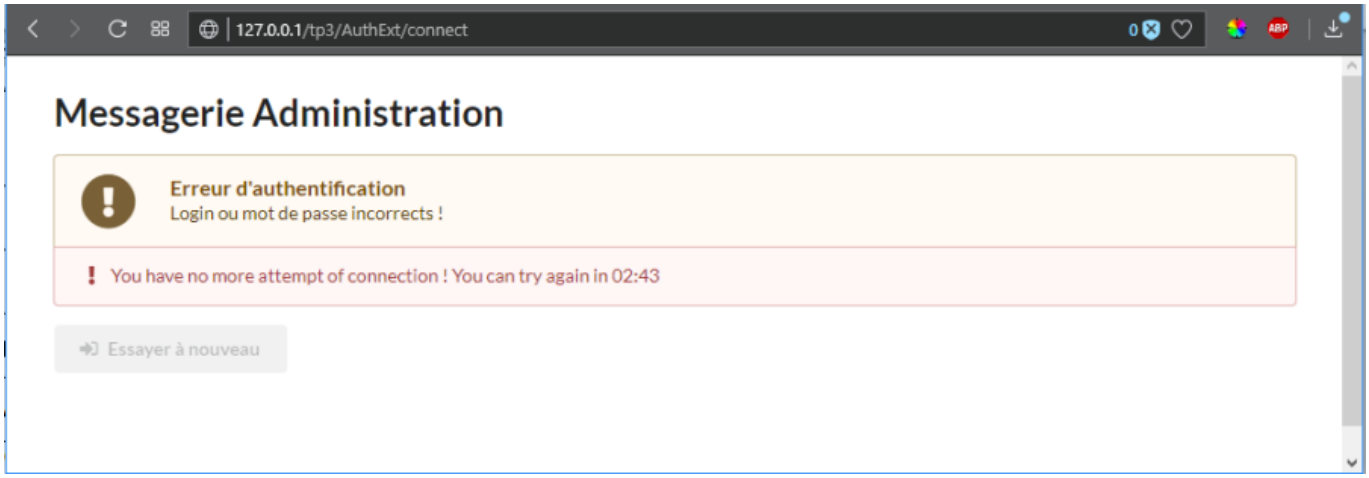
Limiter le nombre de connexions consécutives aboutissant à un échec peut permettre d'éviter les botnets et attaques de force brute.

Surdéfinir la méthode **attemptsNumber** de la classe **AuthExt** : **attemptsNumber** définit le nombre d'erreurs de connexion consécutives autorisées.

```
class AuthExt extends \controllers\BaseAuth{
...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::attemptsNumber()
     */
    protected function attemptsNumber() {
        return 3;
    }
...
}
```

Pour tester :

- Se connecter 3 fois à l'application avec des identifiants inexistants



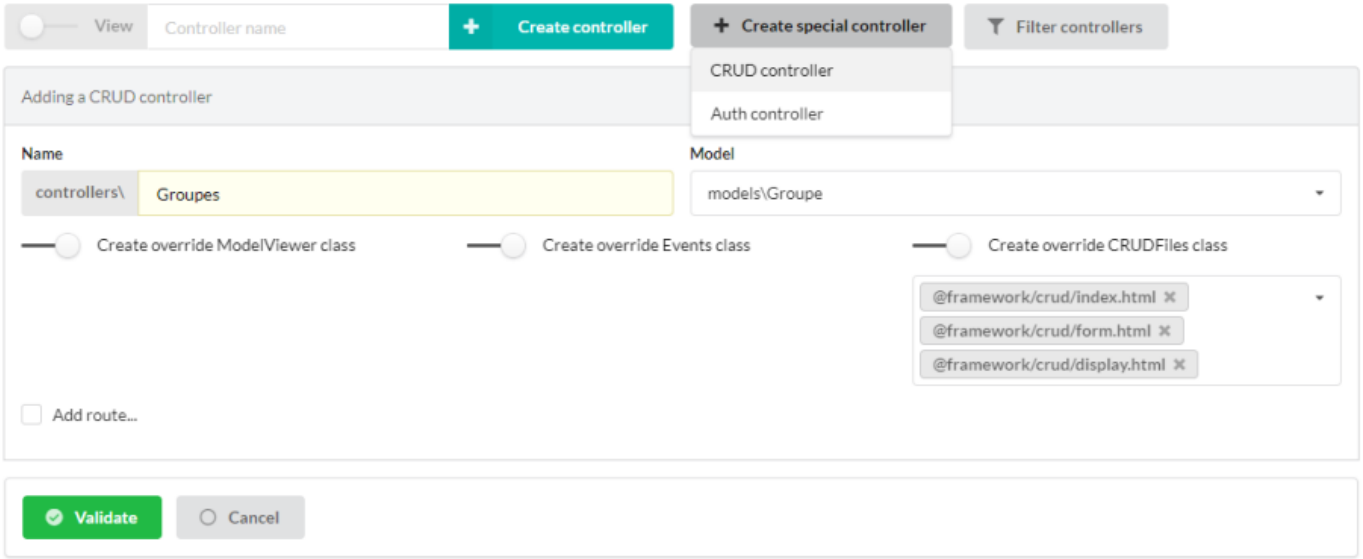
CRUD controllers

Les CRUD controllers permettent d'implémenter rapidement les fonctionnalités CRUD sur un model (scaffolding) :

- Create → création d'objets/ajout d'enregistrement dans la base
- Read → Lecture d'objets/enregistements
- Update → Mise à jour d'objets/enregistements
- Delete → Suppression d'objets/enregistrements

-- Création

A partir de l'interface d'administration, partie **controllers**, choisir **Create special controller/CRUD controller** :



Accéder à l'adresse **/Groupes** ou **/Groupes/index** et tester les opérations CRUD.

-- Intégration

Pour faire en sorte que le contrôleur **Groupes** créé s'intègre à l'application existante, surdéfinir la méthode **getBaseTemplate** de la classe **controllers/crud/files/GroupesFiles**, en choisissant la rubrique

override/implement method du menu source d'Eclipse :

```
namespace controllers\crud\files;

use Ubiquity\controllers\crud\CRUDFiles;
 /**
 * Class GroupesFiles
 **/
class GroupesFiles extends CRUDFiles{
    public function getViewForm(){
        return "Groupes/form.html";
    }
    public function getViewDisplay(){
        return "Groupes/display.html";
    }

    public function getViewIndex() {
        return "Groupes/index.html";
    }
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDFiles::getBaseTemplate()
     */
    public function getBaseTemplate() {
        return "base.html";
    }

}
```

--Authentification

Les contrôleurs CRUD peuvent utiliser l'authentification comme les autres contrôleurs :

Ajouter le trait **WithAuthTrait** dans le contrôleur Groupes, et surdéfinir la méthode **getAuthController** :

```
 /**
 * CRUD Controller Groupes
 **/
class Groupes extends \Ubiquity\controllers\crud\CRUDController{
    use WithAuthTrait{
        initialize as _initializeAuth;
    }

    public function initialize(){
        $this->_initializeAuth();
        if(!URequest::isAjax()){
            $this->loadView("main/vHeader.html");
        }
    }
    public function __construct(){
        parent::__construct();
        $this->model="models\\Groupe";
```

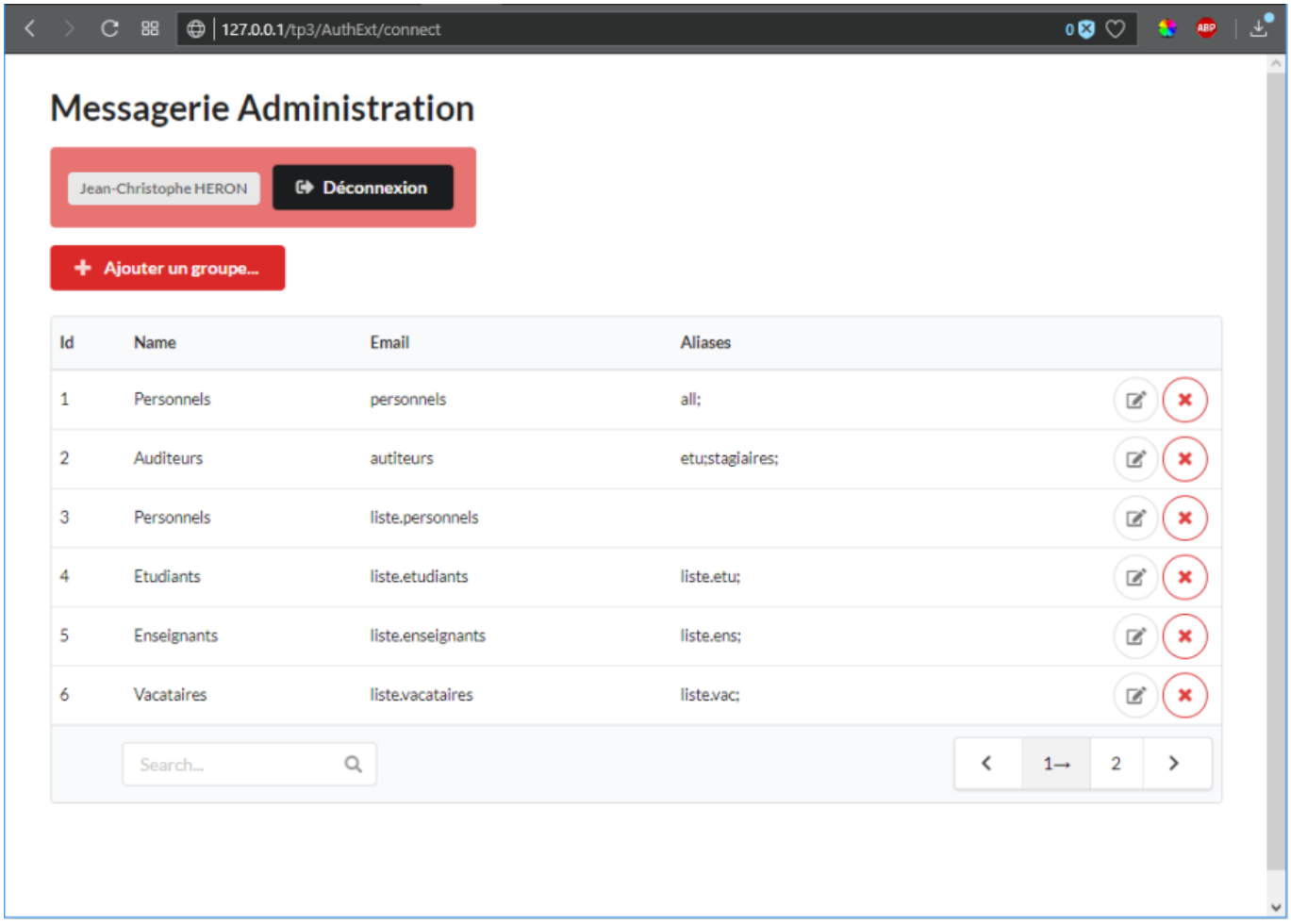
```
    }

    public function _getBaseRoute() {
        return 'Groupes';
    }
    protected function getModelViewer(): ModelViewer{
        return new GroupesViewer($this);
    }

    protected function getEvents(): CRUDEvents{
        return new GroupesEvents();
    }

    protected function getFiles(): CRUDFiles{
        return new GroupesFiles();
    }

    protected function getAuthController(): AuthController {
        return new AuthExt();
    }
}
```

-- Personnalisation de l'affichage

Modifier le template **app/views/Groupes/index.html** pour changer l'apparence du bouton Ajouter, et enlever le segment entourant l'affichage :

```
{% extends "@framework/crud/index.html" %}
{% block _before %}
{% endblock %}
{% block btAddNew %}
    <a id="btAddNew" class="ui button red">
     <i class="icon plus"></i>
    Ajouter un groupe...
    </a>
{% endblock %}
{% block frm %}
    {{ parent() }}
{% endblock %}
{% block dataTable %}
    {{ parent() }}
{% endblock %}
{% block messages %}
    {{ parent() }}
{% endblock %}
{% block details %}
    {{ parent() }}
{% endblock %}
{% block _after %}
{% endblock %}
```

Tester le résultat :



Effectuer un commit+push vers gitHub

-- Personnalisation du contenu et du comportement de la table

Sélection des champs

Pour sélectionner les champs/membres à afficher dans la table, surdéfinir la méthode **getFieldNames** de la classe **controllers/crud/datas/GroupesDatas**

```
class GroupesDatas extends CRUDDatas{
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDDatas::getFieldNames()
     */
    public function getFieldNames($model) {
        return ["name","email"];
    }
}
```

Captions

Pour modifier les captions des champs (en-têtes du tableau), surdéfinir la méthode **getCaptions** de la classe **controllers/crud/viewers/GroupesViewer** :

```
 /**
 * Class GroupesViewer
 **/
class GroupesViewer extends ModelViewer{
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\admin\viewers\ModelViewer::getCaptions()
     */
    public function getCaptions($captions, $className) {
        return ["Nom","Adresse email"];
    }
}
```

Modification du contenu de la table

Pour personnaliser le contenu de la table, surdéfinir la méthode **getDataTableInstance** de la classe **controllers/crud/viewers/GroupesViewer** :

```
 /**
 * Class GroupesViewer
 **/
class GroupesViewer extends ModelViewer{
...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\admin\viewers\ModelViewer::getDataTableInstance()
     */
    protected function getDataTableInstance($instances, $model, $page = 1):
DataTable {
        $dt=parent::getDataTableInstance ($instances,$model,$page);
        $dt->fieldAsLabel(1,"users");
        return $dt;
    }
...
}
```

Modification des actions par ligne

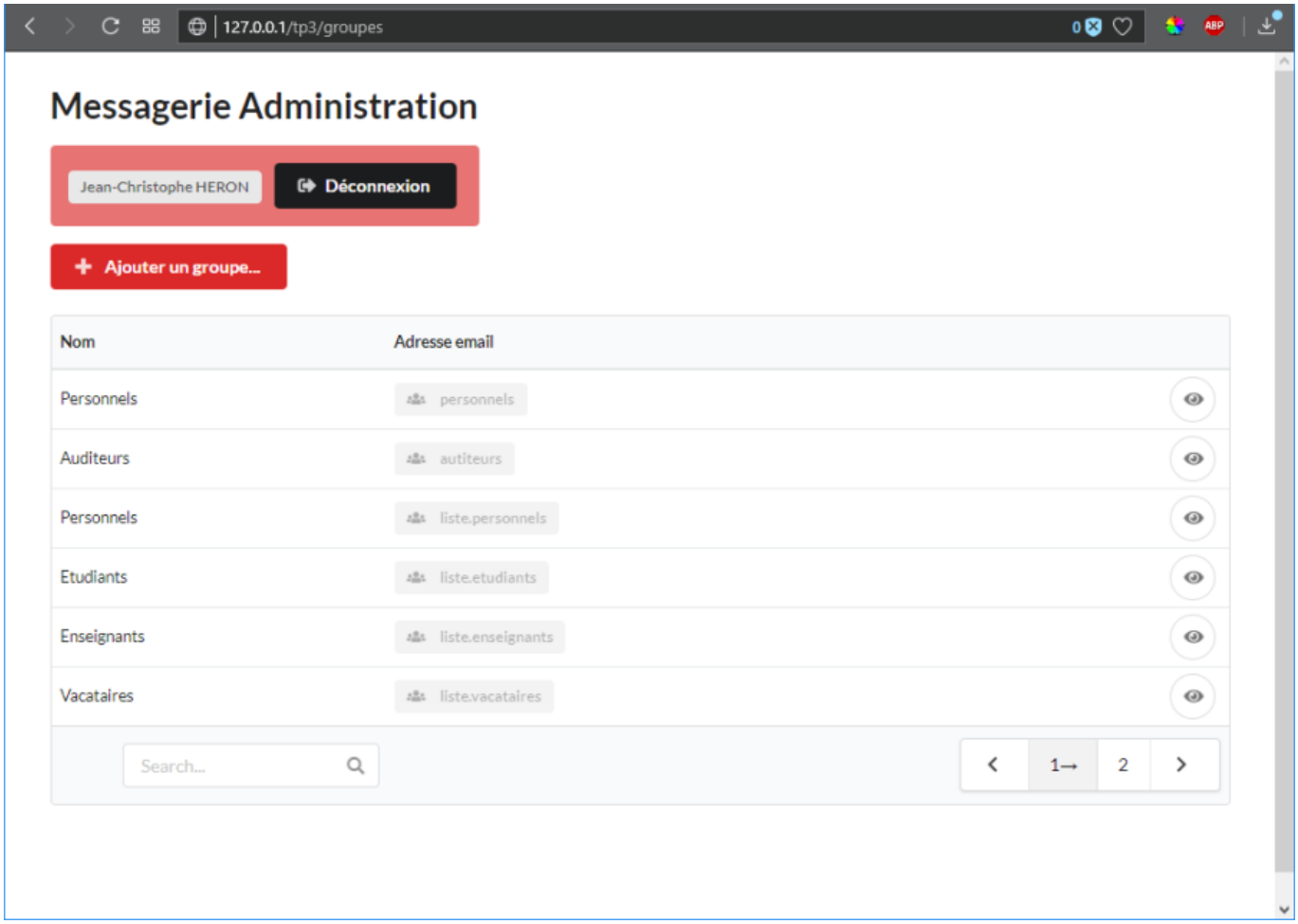
Pour définir les actions à exécuter sur chaque enregistrement (display, edit ou delete), surdéfinir la méthode **getDataTableInstance** de la classe **controllers/crud/viewers/GroupesViewer** :

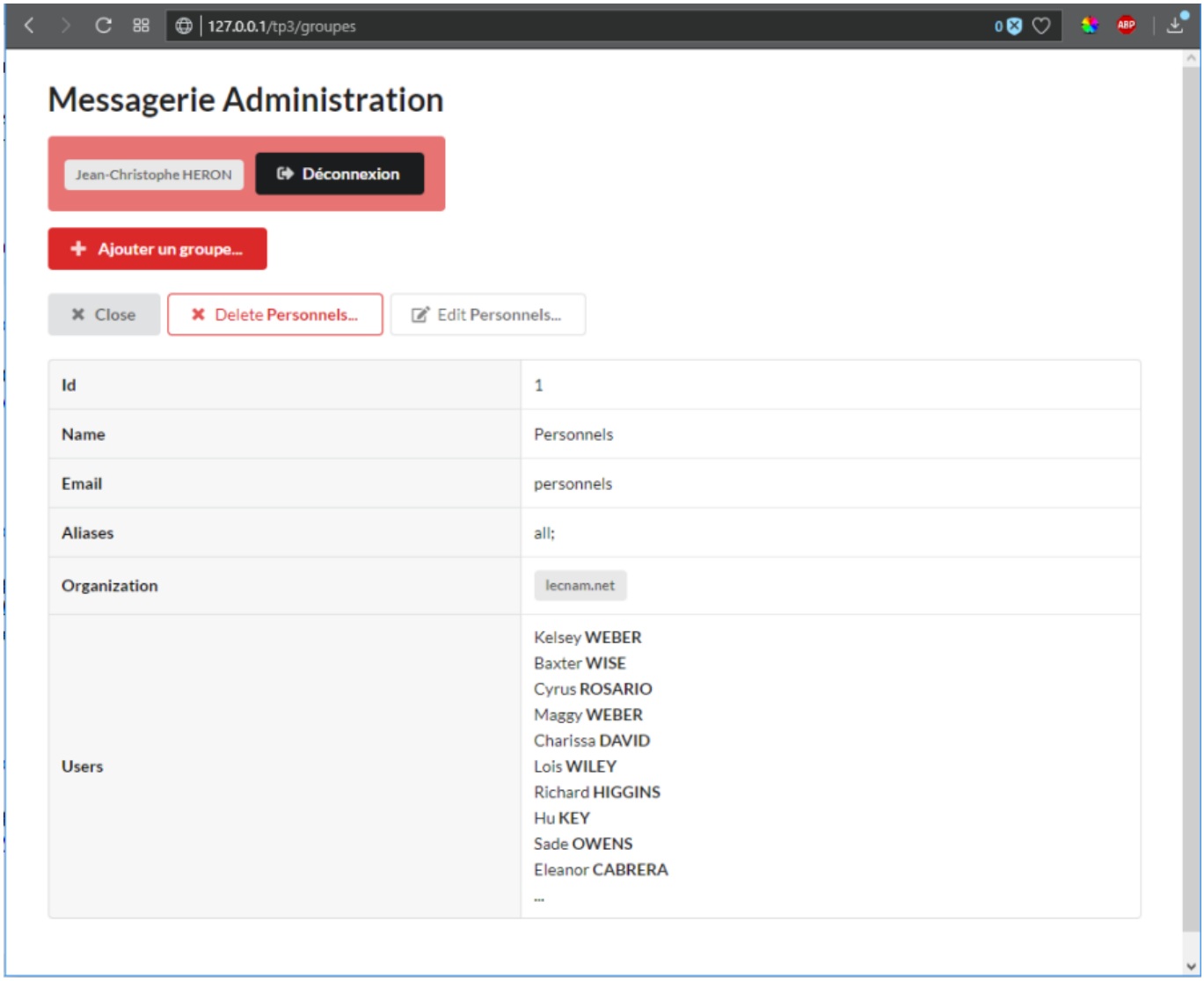
Dans le cas suivant, nous ne laissons que la visualisation d'instance (**display**).

```
 /**
 * Class GroupesViewer
 **/
class GroupesViewer extends ModelViewer{
...
    /**
     * {@inheritDoc}
```

```
     * @see
\Ubiquity\controllers\admin\viewers\ModelViewer::getDataTableRowButtons()
     */
    protected function getDataTableRowButtons() {
        return ["display"];
    }
...
}
```

Tester vos résultats :





Effectuer un commit+push vers gitHub

-- Filtrage des données en fonction de l'authentification

On souhaite afficher les groupes de l'organisation dont fait partie l'utilisateur connecté.

Surdéfinir la méthode **_getInstancesFilter** de la classe **Groupes** : cette méthode retourne la partie **WHERE** d'une instruction SQL.

```
class Groupes extends \Ubiquity\controllers\crud\CRUDController{
...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDController::_getInstancesFilter()
     */
    public function _getInstancesFilter($model) {
        return
"idOrganization=".$this->_getAuthController()->_getActiveUser()->getOrganization()-
>getId();
    }
...
```

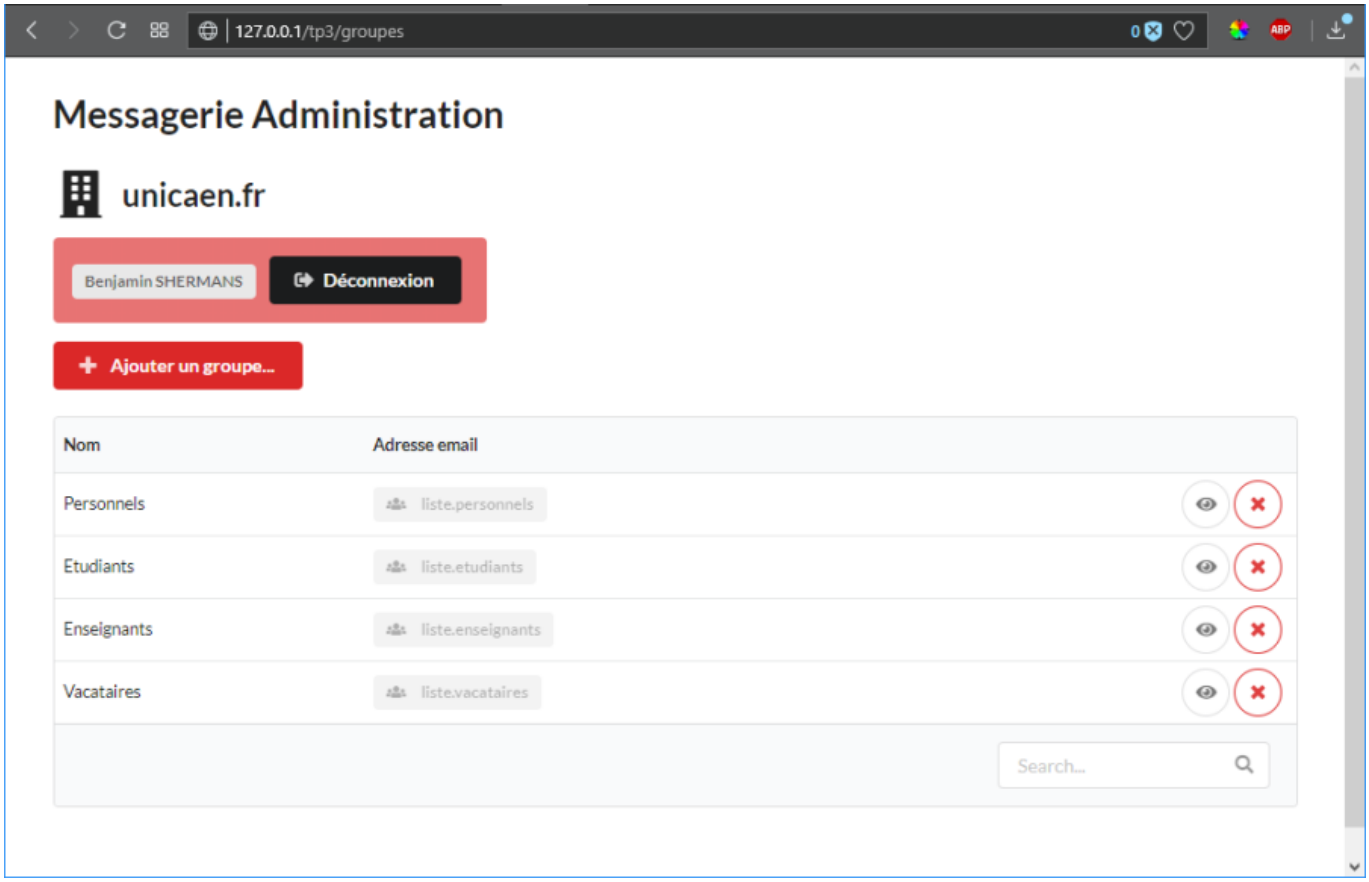
```
}
```

Affichage de l'organisation de l'utilisateur connecté :

Modifier le block **_before** du template **views/AuthExt/info.html** affichant l'information liée à l'utilisateur connecté :

Récupérer l'utilisateur connecté via la variable **{connected}**.

```
{% extends "@framework/auth/info.html" %}
{% block _before %}
<h2 class="ui header">
  <i class="building icon"></i>
  <div class="content">
   {{connected.organization}}
  </div>
</h2>
<div class="ui tertiary inverted red compact segment">
{% endblock %}
{% block _userInfo %}
    {{ parent() }}
{% endblock %}
{% block _logoutButton %}
    {{ parent() }}
{% endblock %}
{% block _logoutCaption %}
    <i class="ui sign out icon"></i>Déconnexion
{% endblock %}
{% block _loginButton %}
    {{ parent() }}
{% endblock %}
{% block _loginCaption %}
    {{ parent() }}
{% endblock %}
{% block _after %}
</div>
{% endblock %}
```



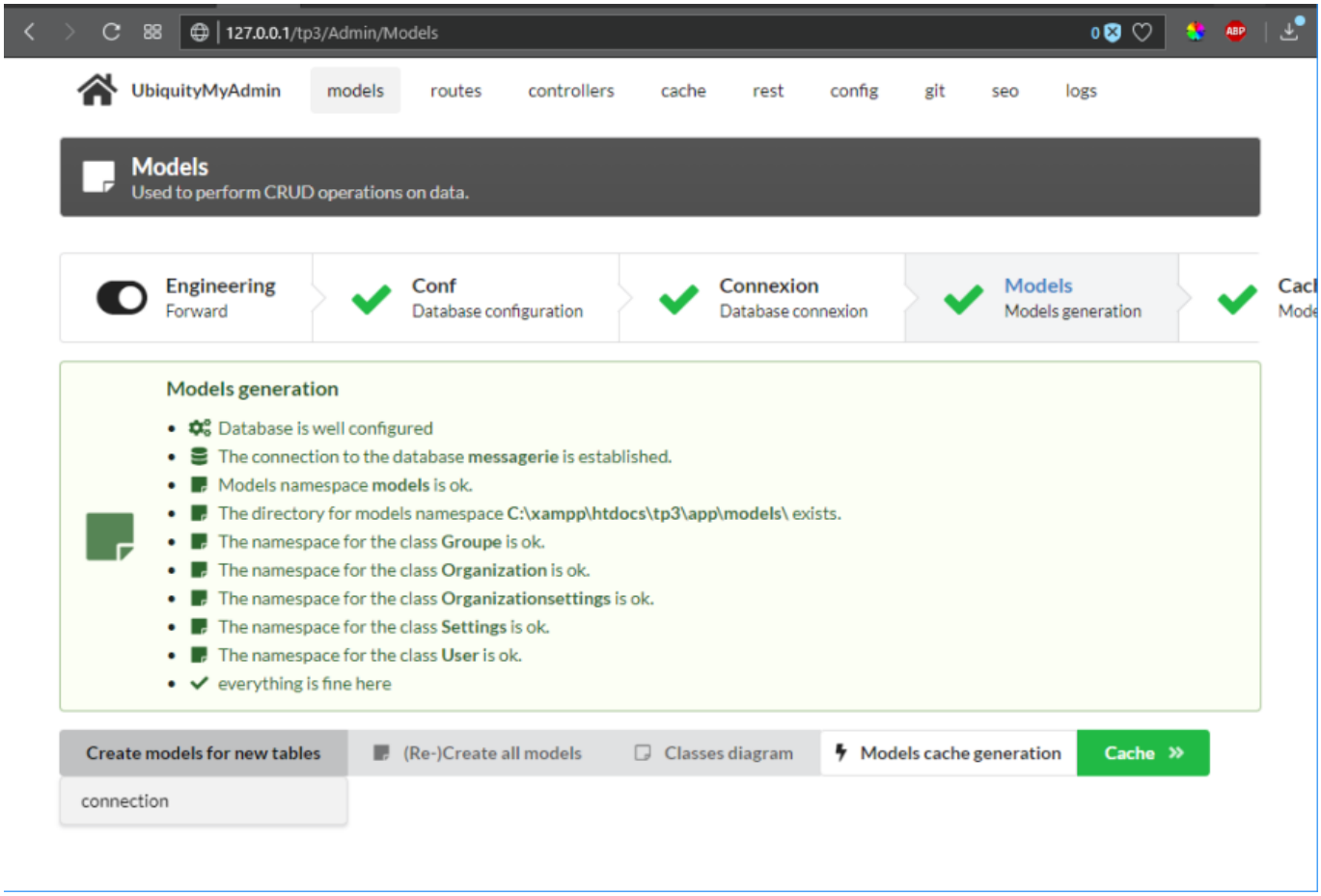
Création de table

Importer et exécuter le script connection.sql depuis phpMyAdmin.

La table **connexion** va permettre d'historiser les connexions à l'application.

-- Génération du model

Depuis l'interface d'administration d'Ubiquity, générer le model correspondant dans la partie **models**.



Initialiser le cache des models comme proposé.

-- Enregistrement de chaque connexion

A chaque connexion d'un utilisateur, on ajoute une instance de connexion, stockée dans la base de données :

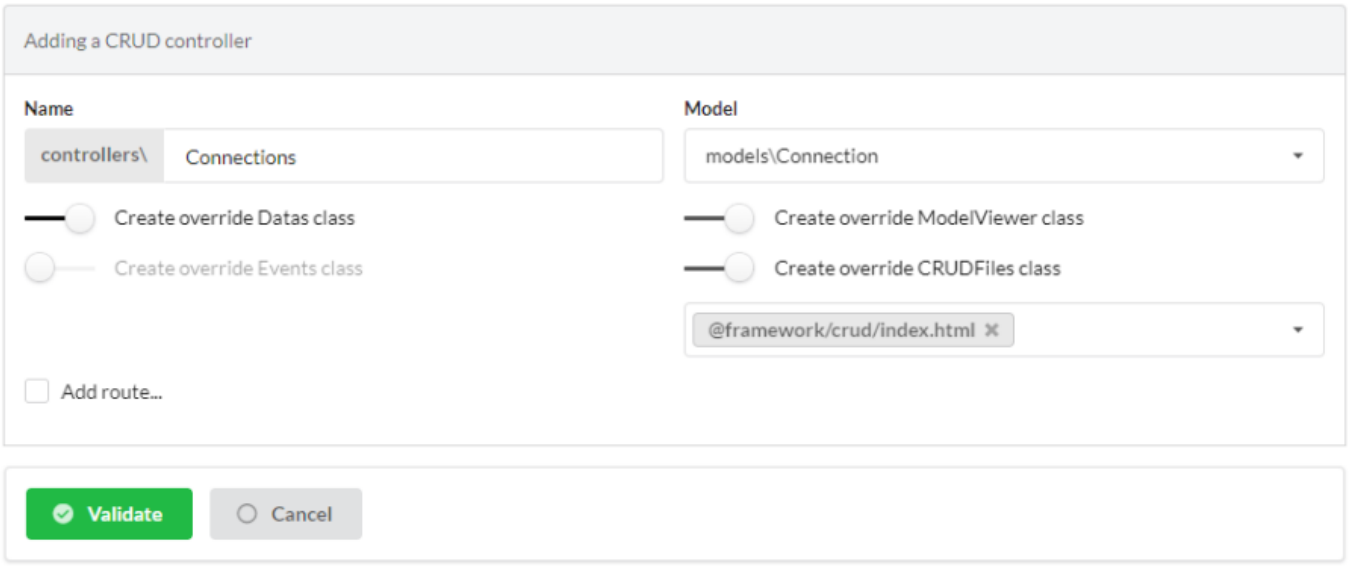
```
class BaseAuth extends \Ubiquity\controllers\auth\AuthController{

    protected function onConnect($connected) {
        $urlParts=$this->getOriginalURL();
        USession::set($this->_getUserSessionKey(), $connected);
        if(isset($urlParts)){
            $url=implode("/",$urlParts);
            Startup::forward($url);
        }else{
            $url="organizations/display/".$connected->getOrganization()->getId();
$this->forward("controllers\\Organizations","display",[$connected->getOrganization(
)->getId()],true,true);
        }
        $connection=new Connection();
        $connection->setUser($connected);
        $connection->setUrl($url);
        $connection->setDateCo(date('Y-m-d H:i:s'));
        DAO::save($connection);
    }
...
```

```
}
```

Tester en vous connectant plusieurs fois à l'application, et en changeant d'utilisateur.
Vérifier ensuite les enregistrements présents dans la table **connection**

-- Crud controller pour les connexions

Créer un CRUD controller pour gérer les connexions :



-- Classement des données

Surdéfinir la méthode **_getInstancesFilter** de la classe **Connections** pour classer les enregistrements par utilisateur et par date :

```
class Connections extends \Ubiquity\controllers\crud\CRUDController{
    public function __construct(){
        parent::__construct();
        $this->model="models\\Connection";
    }

    public function _getBaseRoute() {
        return 'Connections';
    }
    protected function getAdminData(): CRUDDatas{
        return new ConnectionsDatas($this);
    }

    protected function getModelViewer(): ModelViewer{
        return new ConnectionsViewer($this);
    }

    protected function getFiles(): CRUDFiles{
        return new ConnectionsFiles();
    }
```

```
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDController::_getInstancesFilter()
     */
    public function _getInstancesFilter($model) {
        return "1=1 order by idUser DESC,dateCo DESC";
    }
}
```

Sélection des champs à afficher

Surdéfinir la méthode **getFieldNames** de la classe **ConnectionsDatas** pour définir les champs/membres à afficher :

```
class ConnectionsDatas extends CRUDDatas{
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDDatas::getFieldNames()
     */
    public function getFieldNames($model) {
        return ["user","dateCo","url"];
    }
    //use override/implement Methods
}
```

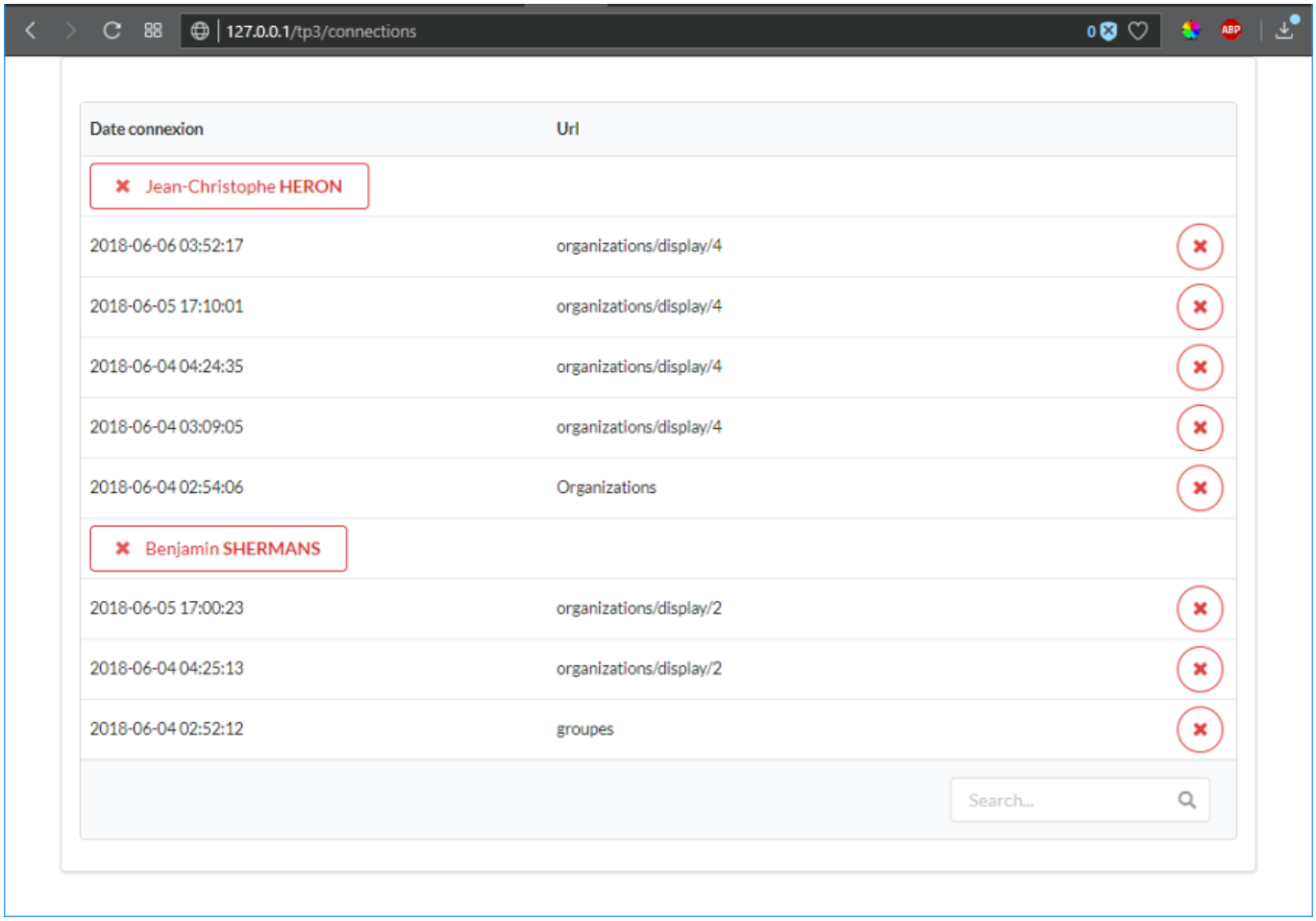
ModelViewer

Surdéfinir les méthodes suivantes de la classe **ConnectionsViewer**, en utilisant la rubrique **override/implement method** du menu **Source** :

```
class ConnectionsViewer extends ModelViewer{
    /**
     * Définition des en-têtes de colonnes
     */
    public function getCaptions($captions, $className) {
        return ["Date connexion","Url"];
    }

    /**
     * Affichage du champ en position 0 (user) sous forme de bouton
     */
    protected function getDataTableInstance($instances, $model, $page = 1):
DataTable {
        $dt= parent::getDataTableInstance ($instances,$model,$page);
        $dt->fieldAsButton(0,"basic
red",["jsCallback"=>function($bt,$instance){$bt->setProperty("data-
user",$instance->getUser()->getId());$bt->addIcon("remove");}]);
        $dt->setEdition ( true );
        return $dt;
    }
```

```
    /**
     * Affichage du seul bouton pour Supprimer
     */
    protected function getDataTableRowButtons() {
        return ["delete"];
    }

    /**
     * Pas d'affichage du détail sur clic d'un élément
     */
    public function showDetailsOnDataTableClick() {
        return false;
    }

    /**
     * Affichage de 5 enregistrements par page
     */
    public function recordsPerPage($model, $totalCount = 0) {
        return 5;
    }

    /**
     * Regroupement suivant le champ en position 0 (user)
     */
    public function getGroupByFields() {
        return [0];
    }
}
```

Tester le résultat à l'adresse **/Connexions** :



Effectuer un commit+push vers gitHub

-- Améliorations de l'affichage

Template index

Modifier le block **_before** du template **views/Connections/index.html** :

- Pour faire apparaître une zone header
- Pour nommer l'élément d'id **zone-co** qui servira par la suite à mettre à jour la table après suppression d'enregistrements

```
{% extends "@framework/crud/index.html" %}
{% block _before %}
    <div class="ui segment" id="zone-co">
    <h2 class="ui header">
        <i class="large icons">
            <i class="sign in icon"></i>
            <i class="inverted corner cogs icon"></i>
        </i>
        <div class="content">
            Administration des connexions
            <div class="sub header">Gestion de l'historique des connexions par
utilisateur</div>
        </div>
```

```
    </h2>
{% endblock %}
...
```

Reprise du template de base

Surdéfinir la méthode **getBaseTemplate** pour reprendre le design de l'application :

```
class ConnectionsFiles extends CRUDFiles{
    public function getViewIndex(){
        return "Connections/index.html";
    }
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\crud\CRUDFiles::getViewBaseTemplate()
     */
    public function getBaseTemplate() {
        return "base.html";
    }
}
```

-- Suppression des enregistrements

La suppression sur le click d'un bouton d'utilisateur se fait en utilisant le helper **_deleteMultiple** présent dans la classe de base **CRUDController** :

```
class Connections extends \Ubiquity\controllers\crud\CRUDController{
    ...
    public function deleteAll($idUser){
        $this->_deleteMultiple($idUser, "deleteAll", "#zone-co",
"idUser=".$idUser);
    }
}
```

L'action **deleteAll** est ensuite associée au click sur chaque bouton user, dans la méthode **onDisplayElements** de la classe **ConnectionsEvents** :

```
class ConnectionsEvents extends CRUDEvents{
    /**
     * Appelé après affichage des objets (refresh ou index)
     * Ajout de l'appel de deleteAll sur le click du bouton user
     */
    public function onDisplayElements() {
        $this->controller->jquery->getOnClick(".ui.button[data-
user]",$this->controller->_getBaseRoute()."/deleteAll/","#table-
messages",["attr"=>"data-user"]);
    }
```

L'attribut html **data-user** présent sur chaque bouton est passé à la méthode **deleteAll** pour permettre de récupérer l'id de l'utilisateur dont on souhaite supprimer les connexions.

Tester la suppression puis effectuer un commit+push vers gitHub

-- Améliorations de l'affichage

Messages de suppression

```
...
class ConnectionsEvents extends CRUDEvents{
    /**
     * Appelé après affichage des objets (refresh ou index)
     * Ajout de l'appel de deleteAll sur le click du bouton user
     */
    public function onDisplayElements() {
        $this->controller->jquery->getOnClick(".ui.button[data-
user]",$this->controller->_getBaseRoute()."/deleteAll/","#table-
messages",["attr"=>"data-user"]);
    }

    /**
     * Modifie le message de confirmation en cas de suppression multiple
     */
    public function
onConfDeleteMultipleMessage(\Ubiquity\controllers\crud\CRUDMessage $message,
$data): CRUDMessage {
        $user=DAO::getOne("models\\User", $data);
        $message->setMessage("Confirmez vous la suppression des connexions de
l'utilisateur `<b>" . $user . "</b>`?");
        $message->setTitle("Confirmation avant suppression");
        return $message;
    }

    /**
     * Modifie le message post suppression multiple
     */
    public function
onSuccessDeleteMultipleMessage(\Ubiquity\controllers\crud\CRUDMessage $message):
CRUDMessage {
        $message->setMessage("Suppression de {count} enregistrement(s)");
        $message->setTitle("Suppression");
        return $message;
    }
}
```

Modification de l'aspect des champs

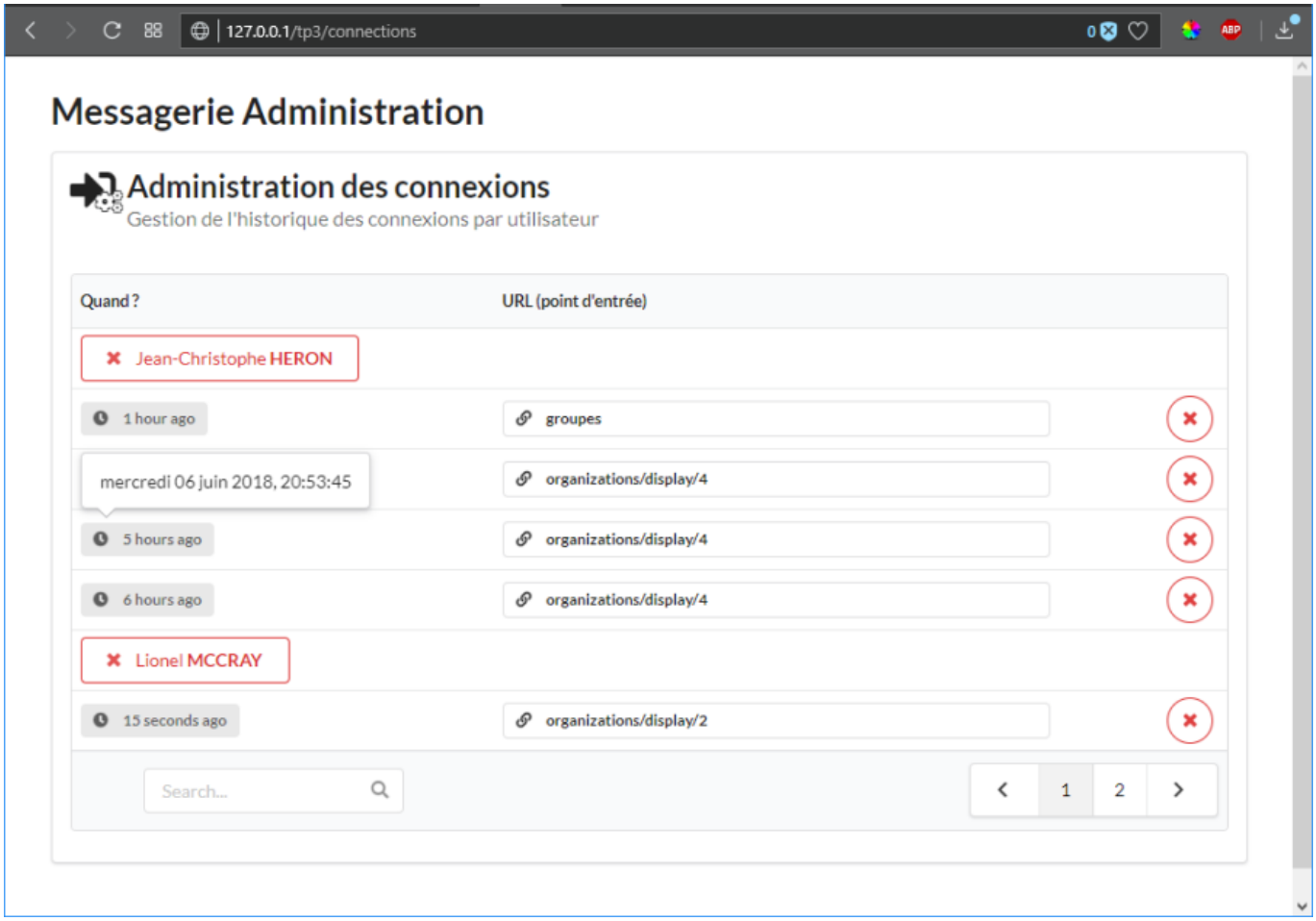
- **dateCo** est affiché dans un label, la date est formatée, le label affiche une popup au passage de la souris
- **Url** est également affiché dans un label

- Les captions des colonnes sont modifiés

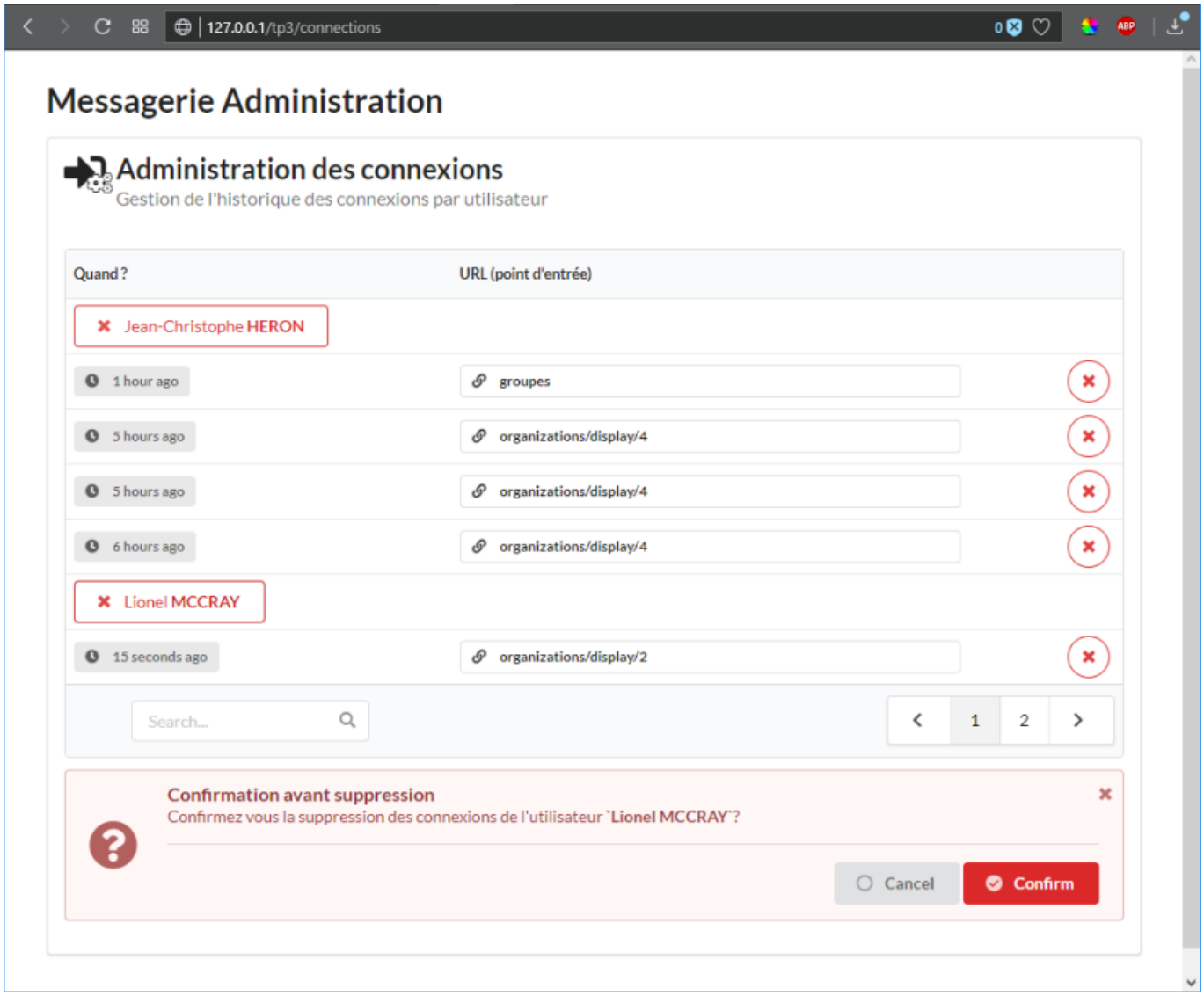
```
class ConnectionsViewer extends ModelViewer{
    /**
     * Définition des en-têtes de colonnes
     */
    public function getCaptions($captions, $className) {
        return ["Quand ?","URL (point d'entrée)"];
    }

    /**
     * Modification de l'affichage des champs
     */
    protected function getDataTableInstance($instances, $model, $page = 1):
DataTable {
        $dt= parent::getDataTableInstance ($instances,$model,$page);
        $dt->fieldAsButton(0,"basic red",["jsCallback"=>function($bt,$instance){
            $bt->setProperty("data-user",$instance->getUser()->getId());
            $bt->addIcon("remove");
        }]);
        $dt->fieldAsLabel(2,'linkify',["addClass"=>"basic fluid"]);
        $dt->setValueFunction(1,function($value,$instance){
            $lbl=new HtmlLabel("dateCo-
".$instance->getId(),UDateTime::elapsed($value),"clock");
            $lbl->addPopup("",UDateTime::longDatetime($value,"fr"));
            return $lbl;
        });
        $dt->setEdition ( true );
        return $dt;
    }
...
}
```

Tester les pages puis effectuer un commit+push vers gitHub

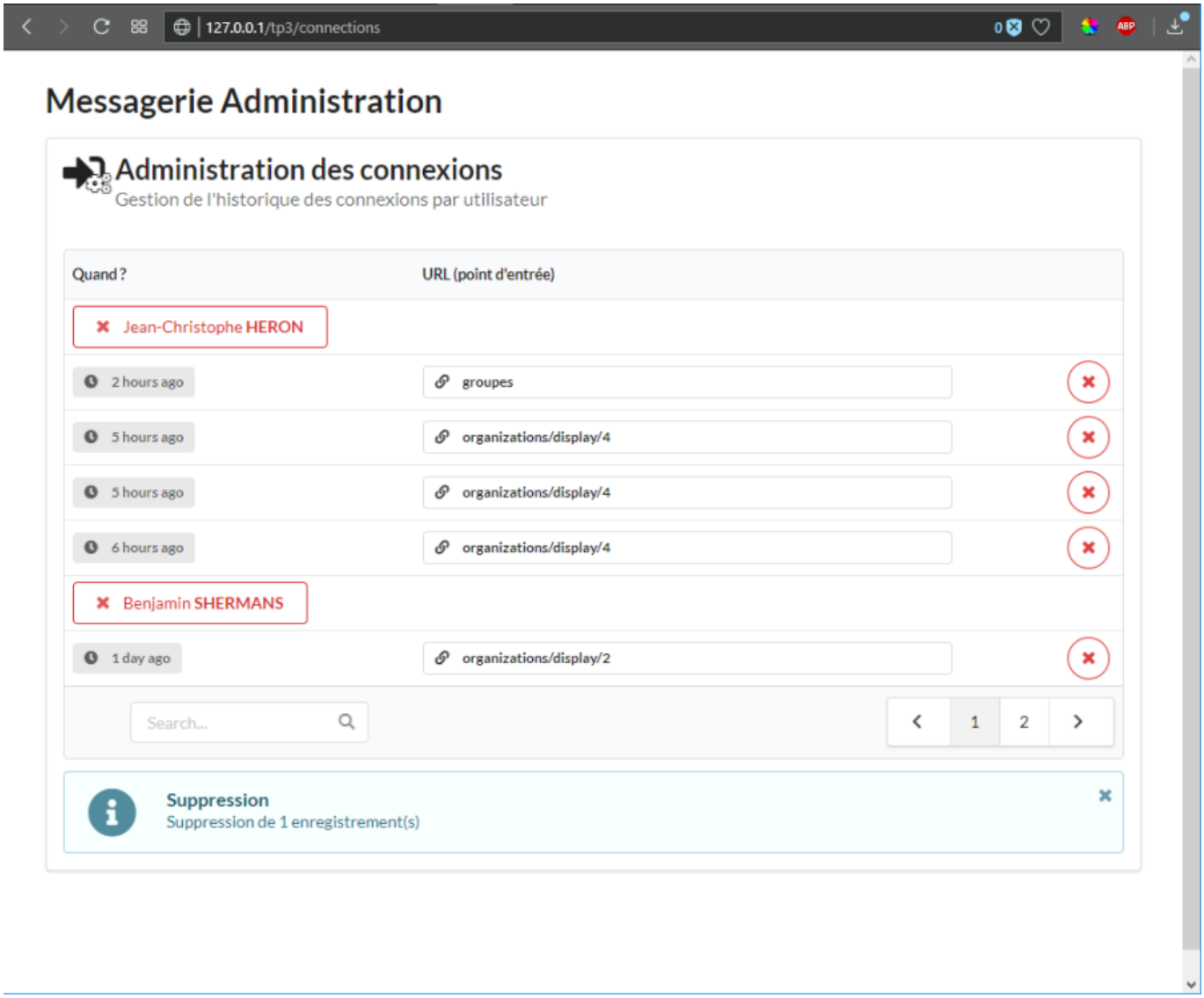
Affichage des connexions



Confirmation avant suppression multiple



Affichage du message de suppression



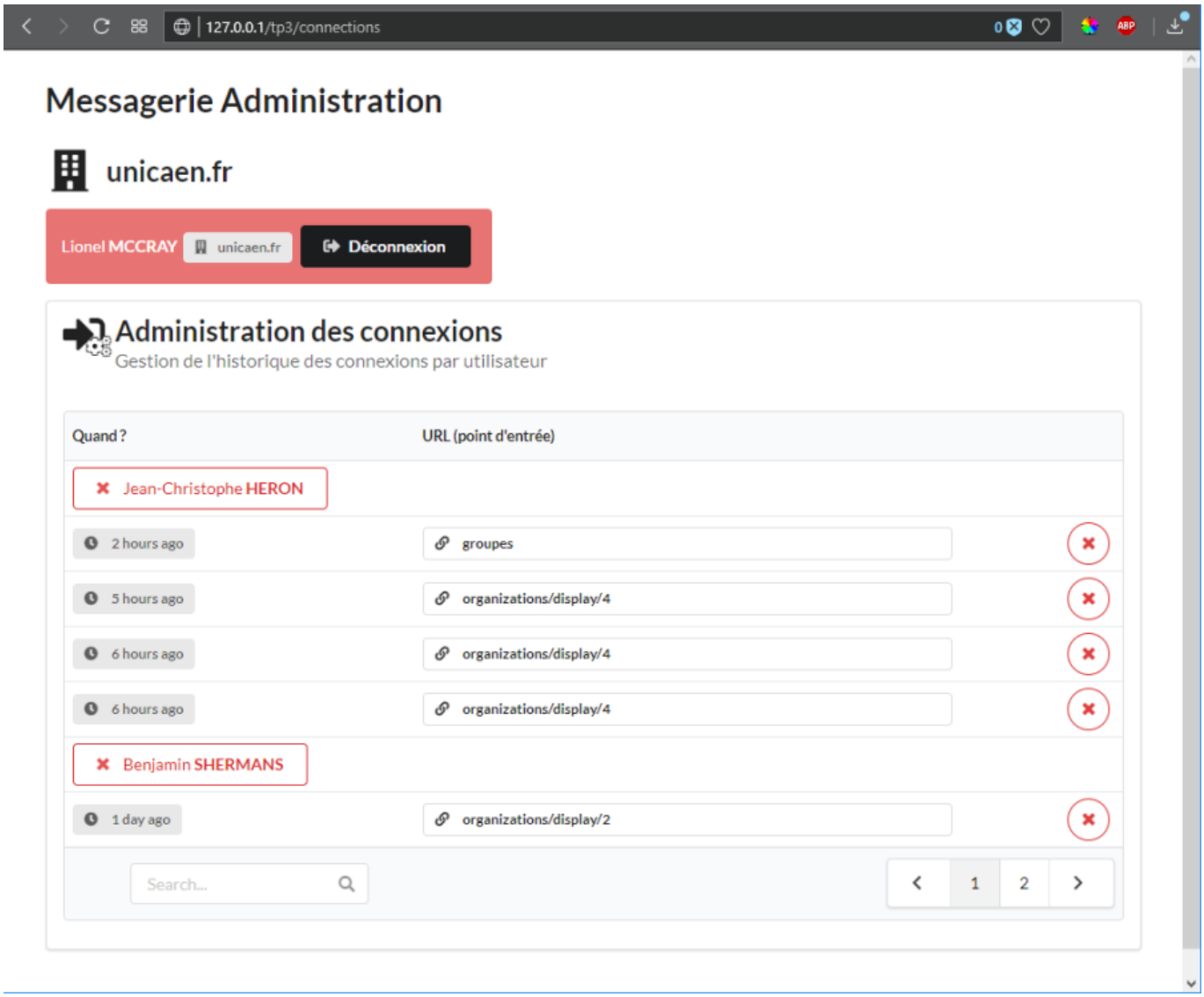
-- Ajout de l'authentification

Ajouter l'authentification via **AuthExt** au controller **Connexions** :

```
...
class Connections extends \Ubiquity\controllers\crud\CRUDController{
    use WithAuthTrait{
        initialize as _initializeAuth;
    }
    public function initialize(){
        $this->_initializeAuth();
        if(!URequest::isAjax()){
            $this->loadView("main/vHeader.html");
        }
    }

    protected function getAuthController(): AuthController {
        return new AuthExt();
    }
```

```
...
}
```

Tester les pages puis effectuer un commit+push vers gitHub



From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/php-rt/tp4?rev=1528335796

Last update: **2019/08/31 14:26**