Collin Maxime

stage CARTAHU

Évolutions

Description des évolutions possibles et éventuellement de la démarche à suivre

Référence du document	LETG_CM_Ev_2-0
Version du document	2.0
Date de version	10/10/14
Etat du document	Fini
Rédacteurs	Maxime Collin
Relecteurs	

Historique des Modifications

Version	Commentaire	Date de modification	Auteur
1.0	Création du document	23/05/14	Maxime Collin
1.1	Ajout évolutions	17/06/14	Maxime Collin
1.2	Ajout évolution et correction	29/08/14	Maxime Collin
2.0	Ajout évolutions	10/10/14	Maxime Collin

Table des matières

I) Sécuriser des URLs en HTTPS

On peut vouloir sécuriser certaines parties du site en utilisant le protocole HTTPS, comme la page de login par exemple. Il faut pour cela demander à un organisme d'autorité de certification de délivrer une accréditation (tel que CAcert). Une autre solution est de générer soi-même ce certificat. Cela est possible par exemple sous linux avec OpenSSL

On installe alors le certificat Serveur de l'autorité de certification qui l'a émis sur le serveur. Sous Apache il faut installer le module ssl et le configurer.

Il reste à imposer aux clients de présenter un certificat Client au serveur. Pour cela il faut rajouter au choix l'une des deux lignes suivantes dans le code source de l'application :

- Dans le fichier app/config/security.yml

```
access_control:
    - { path: /login$, role: IS_AUTHENTICATED_ANONYMOUSLY,
        requires_channel: https }
```

Cette option (*requires_channel*) va forcer le protocole à être en HTTPS, ce qui veut dire qu'on sera redirigé vers une page d'erreur si on essaye d'y accéder via un autre protocole. L'intérêt de cette solution est de pouvoir ainsi protéger toute une zone du site en une seule ligne :

```
access_control:
- { path: /admin, role: ROLE_ADMIN, requires_channel: https }
```

Ainsi toutes les URL comportant /admin ne seront accessibles que via le protocole HTTPS.

- La deuxième solution réécrit l'URL pour utiliser le protocole HTTPS :

  dans un fichier de routing d'un bundle (ici DemoBundle) :

```
secure:
    pattern:  /secure
    defaults: { _controller: AcmeDemoBundle:Main:secure }
    requirements:
        _scheme:  https
```

Cette ligne va vérifier que le protocole est bien HTTPS lorsqu'on tente d'accéder à cette URL et va la réécrire au besoin pour changer le protocole en HTTPS. De cette manière il n'y a pas d'erreur générée. Cependant cela ne protège qu'une seule URL.

Je pense toutefois qu'il est possible de protéger un groupe d'URL en plaçant cette option dans le fichier de routing général (app/config/routing.yml), comme ceci :

```
mc_user:
    resource: "@McUserBundle/Resources/config/routing.yml"
    prefix:   /{_locale}/admin
    requirements:
        _locale: en|fr
        _scheme:  https
```

Cela va alors protéger toutes les routes du fichier src/Mc/UserBundle/Ressources/config/routing.yml

II) Changer le protocole de cryptage des mot de passe

Le protocole utilisé actuellement pour encrypter les mot de passe avant l'enregistrement en base de données est SHA-512. On peut vouloir en utiliser un autre. Il faut pour cela modifier une ligne dans le fichier de sécurité (app/config/security.yml) :

```
security:
    encoders:
        Mc\UserBundle\Entity\Account: sha512
```

Cette ligne détermine le protocole d'encryptage, il suffit de la modifier pour placer le protocole désiré. Attention toutefois à vérifier que ce protocole existe bien sous Symfony et le bundle *FOSUserBundle*, il faudra sinon suivre la démarche pour l'installer.

III) Demander une confirmation d'e-mail lors de la création d'un utilisateur

Aucune confirmation n'est demandée pour l'instant lors de la création d'un compte utilisateur. Le compte est immédiatement opérationnel. Cependant on peut vouloir que l'utilisateur active son compte en suivant un lien envoyé par mail. Le bundle *FOSUserBundle* prend tout en charge et il suffit simplement de dé-commenté les lignes suivantes du fichier app/config/config.yml :

```
fos_user:
    registration:
#       confirmation:
#            from_email:
#                address:        webmaster@managechart.org
#                sender_name:    webmaster
#            enabled:    true
#            template:   FOSUserBundle:Registration:email.txt.twig
```

Pour dé-commenté une ligne il suffit de supprimer le '#' en début de ligne. On peut bien sûr personnaliser ces lignes, comme modifier l'adresse ou le nom de l'expéditeur de l'e-mail ou le corps du message. Le corps du message est dans vendor/friendsofsymfony/user-bundle/FOS/UserBundle/Resources/views/Registration/email.txt.twig . Pour l'instant il ressemble à ceci :

```
sujet : Bienvenue %username% !

Corps :
Bonjour %username% !

      Pour valider votre compte utilisateur, merci de vous rendre sur
%confirmationUrl%

      Cordialement,
      L'équipe.
```

Où *%username%* et *%confirmationUrl%* correspondent respectivement au nom de l'utilisateur et au lien de confirmation pour activer le compte.

Pour modifier le message il faut créer un fichier email.txt.twig dans le répertoire app/Resources/FOSUserBundle/views/Registration. Ce fichier surchargera le précédent et sera sélectionné à la place. Prendre exemple sur le premier fichier et d'autres fichiers .twig du répertoire app/Resources/FOSUserBundle/views pour construire le nouveau fichier.

IV) N'autoriser que le *SELECT* sur les bases de données distantes

Il faudrait s'assurer qu'on ne peut faire que un *SELECT* sur une base de données distantes. Normalement les comptes saisis n'ont que les droits en lecture seule, mais il peut être bien de n'autoriser que les requête de lecture. Actuellement pour plus de simplicité, la requête n'est pas vérifiée, elle est directement exécutée. La vérification devrait être fait au niveau du Bundle *BddBundle*, soit par une lecture de la requête SQL dans la classe abstraite qui implémente l'interface *InterfaceBDD*, soit en utilisant que des fonctions de *SELECT* des extensions PHP dans les classes filles (*ex : pg_select()*). Cette deuxième méthode est plus sûre mais il faut s'assurer que toutes les extensions PHP ont une telle fonction et cela implique certainement de modifier la construction de la requête SQL dans le formulaire *DataList* (et peut-être la base de données) ou de décortiquer cette requête pour exécuter la fonction avec les bons paramètres.

V) Verrouiller un graphique

Une amélioration envisageable est de laisser à un administrateur la possibilité de verrouiller un graphique, empêchant toute modification ou suppression du graphique tant qu'un administrateur n'a pas déverrouillé le graphique. Cela se ferait avec une case à cocher dans la vue qui affiche la liste des graphiques ou dans celle qui affiche les détails d'un graphique ou plus probablement dans les deux. Cette case à cocher n'est affiché bien entendu que si l'utilisateur est administrateur. Elle déclencherait une requête Ajax sur une URL protégée par le préfixe admin/ . Par la suite il faudrait vérifier dans la méthode d'édition et de suppression si cette case à cocher est à true, et générer une erreur si tel est le cas. Enfin il faut bien entendu ajouter un attribut à l'entité *Chart* pour stocker cette valeur.

VI) Conserver le password lors de l'édition d'une DataSource

Pour l'instant lorsqu'on édite une DataSource, le mot de passe est supprimé par défaut. C'est le comportement de Symfony pour plus de sécurité. On peut vouloir le conserver. Une solution à laquelle je pense est de simplement le réaffecter avec son setter avant d'afficher le formulaire.

VII) Changer la locale des BDD distantes

Actuellement lorsqu'on reçoit des données issues de fonctions SQL produisant des chaînes de caractères (ex : *MONTHNAME* qui renvoi le nom du mois de la date), ils sont en anglais. Il peut être intéressant de modifier la locale dans laquelle on récupère ces données pour s'adapter à la locale courante de ManageChart. Pour cela il faut exécuter une requête SQL avant d'exécuter le *SELECT*, or tel que c'est implémenté pour l'instant on ne peut exécuter qu'une seule requête dans l'interface *DataList*.

Je vois 2 solutions à cette évolution :

- Modifier la fonction des drivers MySQL, Postrgres... pour qu'elle puisse exécuter plusieurs fonctions en une fois.
- Exécuter systématiquement le changement de locale avant toute requête. Pour cela il faut récupérer la locale qui est dans l'URL, la passer à l'entité *DataList* avec un paramètre facultatif qui a comme valeur par défaut la locale par défaut de ManageChart, pour enfin la passer à la fonction *query()* du driver. Là cette fonction exécute d'abord le changement de locale avec la bonne valeur, puis exécute la requête. Le changement de locale aura probablement sa place dans la fonction *settings()*.

VIII) Ajout d'un bouton reconstruction graphique

Lorsqu'on est en édition/création d'un graphique, il y a des champs qui ne se mettent pas à jour tel que la légende ou les crédits. Cela vient du fait que ces options doivent être défini à la création du graphique Highchart. Pour les mettre à jour il y a deux solutions :

- Recharger la page, mais on perd tous les formulaires d'axes des ordonnées et les séries.
- Créer un bouton de reconstruction du graphique. Ce bouton reprendrait le principe de l'édition, il pourrait même peut-être enregistrer le graphique tel quel (mais dans ce cas il faut qu'il soit valide), pour le ré-éditer aussitôt. L'autre solution est d'enregistrer toutes les données et de tout reconstruire. Toutefois cela peut prendre du temps.

IX) Gestion des erreurs lors de la soumission d'un formulaire

Actuellement lorsqu'un formulaire est soumis avec des erreurs, Symfony le ré-affiche avec les erreurs et les données. C'est le comportement désiré. Toutefois sur les formulaires où l'application prend la main sur l'affichage par défaut de Symfony, cela n'est plus viable. En effet ce serait le formulaire par défaut qui serait affiché. Il faudrait donc gérer cela lors du test dans le contrôleur sur la validation du formulaire.