ELIE
Benoît

Épreuve SLAM5

Sommaire

1) Introduction

L'épreuve consiste à implémenter une nouvelle fonctionnalité sur un existant.

Il est demandé de conserver les trace des modification, d'anticiper et de prévoir ses répercutions.

La mise à jour des documents (ou tout du moins les modification à apporter doivent être signalés.

2) Mission tel que demandé

On souhaite mettre en place une procédure d'authentification. Chaque utilisateur doit disposer d'un login et mot de passe stockés en base de données.

Après authentification, la page d'accueil de l'étudiant listera les 7 derniers stages saisis en base de données.

Après authentification, la page d'accueil de l'enseignant listera les dates de stages de chaque section et chaque promo de l'année scolaire en cours (page qui sera développée par un de vos collègues).

En cas d'échec, le formulaire d'authentification sera affiché de nouveau.

Dans la première phase de l'épreuve, vous êtes chargé(e) d'étudier cette évolution et de proposer l'ensemble des modifications à envisager pour la réaliser. Les IHMs à ajouter ou modifier seront maquettées.

Dans la deuxième phase, vous réaliserez les modifications de l'application en documentant toutes vos actions.

Chaque phase sera documentée de façon structurée et méthodique.

3) Reformulation de la mission

Il est donc demandé :

- d'ajouter une couche de protection par compte via un couple identifiant/mot de passe,
- pour l'étudiant, changer la page d'accueil pour les 7 derniers stages saisie en BDD,
- pour l'enseignant, changer la page d'accueil pour une page affichant les périodes de stage, par section, pour l'année en cours,
- à l'échec d'authentification, on ré-affiche le formulaire.

MAJ : La gestion étudiant/enseignant impliqué aussi l'attribution de rôle lors de la gestion des comptes, ce point aillent était oublié dans un premier temps, il n'est donc pas implémenté.

D'un point de vue méthodologique, vous devez :

- proposer une modification qui supporte les point précédemment cité,
- réaliser les interfaces Homme-Machine,

- réaliser la modification,
- documenter la modification.

4) Premières actions

Afin de réaliser correctement la mission, j'ai commencé par réaliser :

- le document de synthèse ci-présent,
- la reformulation de la problématique,
- la mise en place d'une branche de développement dédié GIT,
- une synchronisation de cette branche sur GitHub,
- reprise du projet existant.

5) Réflexion de mise en place

Le point clef de cette modification repose sur la mise en place d'un portail captif sur l'application, chargé de demander une authentification à l'utilisateur.

Ce portail doit donc être composé des éléments technique suivants :

- un formulaire de connexion,
- une entité représentant les utilisateur en bdd,
- un controller pour se connecter,
- du code chargé de vérifier si l'utilisateur est enregistré ou non (et le rediriger vers le controller de connexion.

Les trois premiers point serons groupé dans un bundle « connexion », et la portion de code chargé de la vérification sera implémenté dans un premier temps sur toutes les fonction de l'application (méthode lourde), mais une solution en remontant dans les couches (notamment au niveau du constructeur du bundle) simplifierais les opérations.

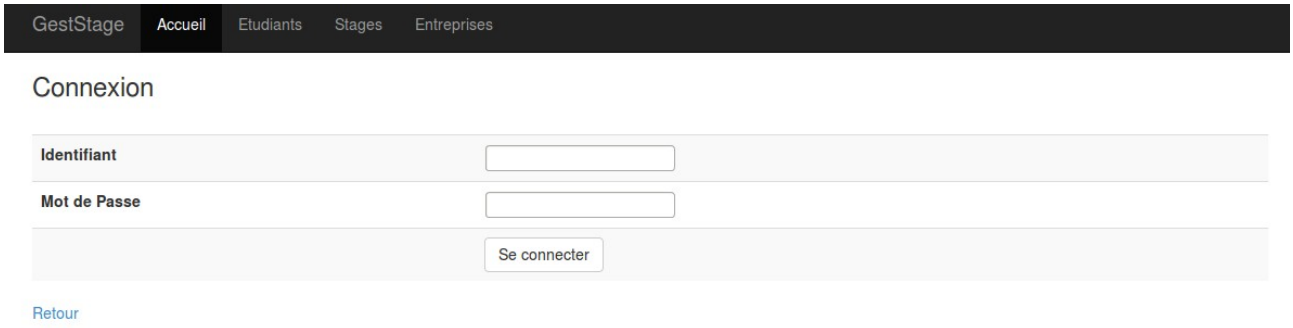
MAJ : Après recherche, l'utilisation de constructeur personnalisé est possible et sera donc retenu.

Il manque aussi la persistance de $membre en $_SESSION pour pouvoir tester l'authentification sur le reste de l'application.

6) Conception des interfaces

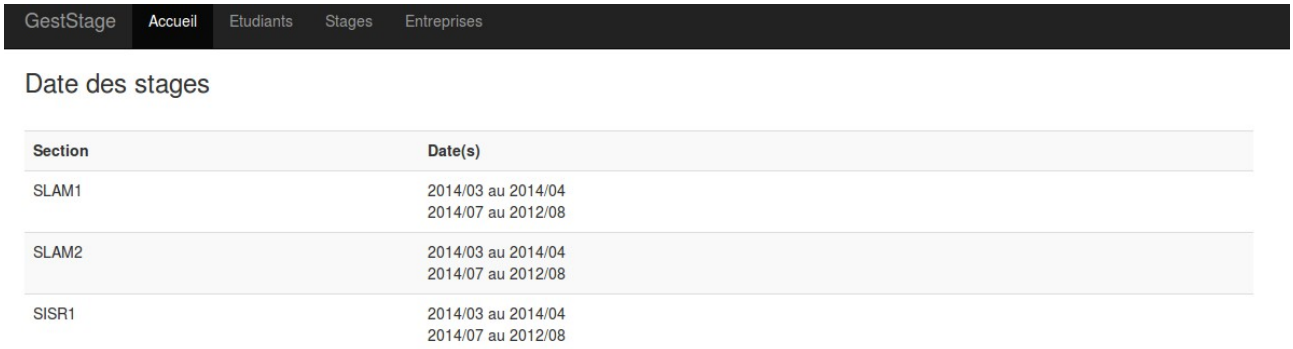
6.1) De connexions

L'interface utilisé sera conforme à la maquette suivante :



6.2) LISTE DES DATES

Maquette :



6.3) DE LISTE DES STAGE DE L'ÉTUDIANT

L'application disposent déjà d'une telle interface, seul le nombre de résultat à affiché sera modifier.

7) MODIFICATION DE LA DOCUMENTATION

Il doit être modifier au terme de l'opération :

- Le document récapitulatif des interfaces,
- le diagramme de navigation,
- le diagramme UML,
- le document de test.

8) Modification effective de l'application

8.1) Mise en place du Bundle

```
Each bundle is hosted under a namespace (like Acme/Bundle/BlogBundle).
The namespace should begin with a "vendor" name like your company name, your
project name, or your client name, followed by one or more optional category
sub-namespaces, and it should end with the bundle name itself
(which must have Bundle as a suffix).

See http://symfony.com/doc/current/cookbook/bundles/best_practices.html#index-1 for more
details on bundle naming conventions.

Use / instead of \  for the namespace delimiter to avoid any problem.

Bundle namespace:webStudent/ConnexionBundle

In your code, a bundle is often referenced by its name. It can be the
concatenation of all namespace parts but it's really up to you to come
up with a unique name (a good practice is to start with the vendor name).
Based on the namespace, we suggest webStudentConnexionBundle.

Bundle name [webStudentConnexionBundle]:

The bundle can be generated anywhere. The suggested default directory uses
the standard conventions.

Target directory [/media/hp_16go/cours/slam5/curent/src]:

Determine the format to use for the generated configuration.

Configuration format (yml, xml, php, or annotation): yml

To help you get started faster, the command can generate some
code snippets for you.

Do you want to generate the whole directory structure [no]? yes

  Summary before generation

You are going to generate a "webStudent\ConnexionBundle\webStudentConnexionBundle" bundle
in "/media/hp_16go/cours/slam5/curent/src/" using the "yml" format.

Do you confirm generation [yes]?

  Bundle generation

Generating the bundle code: OK
Checking that the bundle is autoloaded: OK
Confirm automatic update of your Kernel [yes]?
Enabling the bundle inside the Kernel: OK
Confirm automatic update of the Routing [yes]?
Importing the bundle routing resource: OK
```

8.2) Mise à jour du routing

```
Connexion_accueil:
    path:     /connexion
    defaults: { _controller: webStudentConnexionBundle:Connexion:index }

```

8.3) Création de la vue de connexion

```
{% extends "layout.html.twig" %}

{% block content %}

<h3>Connexion</h3>
<br/>
<form role="form" method="post" {{ form_enctype(form) }}>
    <table class="table table-striped">
        <tr>
            <th>Identifiant</th>
            <td>{{ form_widget(form.login) }}</td>
        </tr>
        <tr>
            <th>Mot de passe</th>
            <td>{{ form_widget(form.password) }}</td>
        </tr>

        <tr>
            <th></th>
            <td><input type="submit" class="btn btn-default" value="valider" /></td>
        </tr>
    </table>
    {{ form_rest(form) }}
</form>
{% endblock %}
```

8.4) Génération de l'entity

```
The Entity shortcut name: webStudentConnexionBundle:Membre

Determine the format to use for the mapping information.

Configuration format (yml, xml, php, or annotation) [annotation]:

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, blob, guid.

New field name (press <return> to stop adding fields): login
Field type [string]:
Field length [255]: 150

New field name (press <return> to stop adding fields): password
Field type [string]:
Field length [255]: 150

New field name (press <return> to stop adding fields):

Do you want to generate an empty repository class [no]? yes

  Summary before generation

You are going to generate a "webStudentConnexionBundle:Membre" Doctrine2 entity
using the "annotation" format.

Do you confirm generation [yes]?

  Entity generation

Generating the entity code: OK
```

8.5) Création du formulaire type

```php
<?php
namespace webStudent\ConnexionBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class ConnexionType extends AbstractType
{
        /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('login')
            ->add('password')
        ;
    }

    /**
     * @param OptionsResolverInterface $resolver
     */
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'webStudent\ConnexionBundle\Entity\Membre'
        ));
    }

    /**
     * @return string
     */
    public function getName()
    {
        return 'webstudent_connexionbundle_connexion';
    }
```

8.6) Mise en place de la fonction de login de connexion

```php
<?php

namespace webStudent\ConnexionBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;
use webStudent\ConnexionBundle\Entity\Membre;
use webStudent\ConnexionBundle\Form\ConnexionType;

class ConnexionController extends Controller
{
    public function indexAction()
    {
        $membre = new Membre ();
        $form = $this->createForm(new ConnexionType, $membre);

        // 1. On récupère la requête HTTP
        $request = $this->get('request');

        // 2. On vérifie qu'elle est de type POST
        if ($request->getMethod() == 'POST') {
            // 3. On fait le lien Requête <-> Formulaire
            $form->bind($request);

            // On vérifie que les valeurs entrées sont correctes
            if ($form->isValid()) {
                // on va chercher en BDD si on a un membre
                $odbMembre = $this->getDoctrine()
                    ->getManager()
                    ->getRepository('webStudentConnexionBundle:Membre');

                // On récupère l'entité correspondant à le login $login
                $leMembre = $odbMembre->findOneBy(
                    array(
                        'login'=>$membre->getLogin(),
                        'password'=>$membre->getPassword(),
                        );

                // si on a un membre valide on redirige
                if($membre !== null){
                    return $this->redirect($this->generateUrl('Stage_accueil'));
                }
            }
        }

        // si on a pas quite la fonction avant, on affiche le form de connexion
        return $this->render('webStudentConnexionBundle:Connexion:connexion.html.twig',
            array(
                'form' => $form->createView(),
            ) );
    }

}
```

MAJ : À la ligne 41 il serai possible de tester le rôle contenue dans $membre afin de faire une redirection différente suivant.

Il manque aussi la persistance de $membre en $_SESSION pour pouvoir tester l'authentification sur le reste de l'application.

9) Tests à réaliser

Il conviendra de réaliser les test suivant afin de juger de la bonne intégration :

- refus d'affichage si non connecté/mdp ou login faux et ce peu importe l'url demandé,
- affichage non régressif de l'application une fois non connecté.

10) État actuel de la modification

En l'état, la modification n'est pas fonctionnel.

L'entity est crée, le routing aussi, le formulaire type de même.

Il reste :

- vérifier la fonction de gestion des login (connexionController :index) et ajouter une persistance de l'objet membre dans $_SESSION.
- création et implémentation de la fonction de vérification à placer dans les constructeur.

11) Difficultés rencontrées

11.1) Gestion du traitement interne de symfony inconnus

Le fonctionnement interne exacte n'étant pas connus lors de la réflexion, la possibilité de ne pas mettre en place la solution la plus adapté semble forte.

Dans le cas présent, la possibilité d'inclure un constructeur personnel afin de gérer le portail captif au niveau des bundle est inconnus. Après recherche elle est possible et sera donc retenu.

11.2) Traitement de la déconnexion

Le sujet ne précisant pas si les interfaces et les fonction de gestion de la déconnexion devais être incorporé, il a étais assumé qu'elles étais hors du champ de ce sujet.

11.3) Oublis de la gestion des droits

La table Membre ne gère pas le type d'accès dont dispose le membre.

De sorte qu'il n'est pas possible de différencier les étudiant des prof, ce qui empêche les routage différent suivant le rôle.