Developpez.com Club des développeurs

Tutoriel de création d'un Blog avec PRADO

par Pradosoft Eric Marchetti (traducteur)

Date de publication : 05-09-2007

Dernière mise à jour :

Le but de ce tutoriel est de fournir aux nouveaux utilisateurs de PRADO, un guide pas à pas sur la façon de développer avec PRADO.

Traduction de : **PRADO Blog Tutorial**

I - Introduction..... 4
I-A - Bienvenue dans ce tutoriel de création d'un Blog..... 4
I-B - Buts à atteindre..... 4
Gestion des utilisateurs..... 4
Gestion des messages..... 4
Maintenance de l'outil..... 4
II - 1er jour : Découvrir PRADO..... 5
II-A - Installation..... 5
Les fichiers initiaux..... 5
Les dossiers initiaux..... 6
Personnalisation..... 7
II-B - Création de la page Contact..... 7
Création de la page gabarit..... 8
Création du fichier de classe PHP..... 10
II-C - Partager les modèles de gabarit..... 12
Création du gabarit principal..... 12
Utilisation du gabarit principal..... 13
Autres possibilités pour spécifier le gabarit principal..... 14
III - 2ième jour: Mise en place de la Base de Données..... 15
III-A - Création de la base..... 15
III-B - Connexion à la base..... 17
III-C - Création des classes Active Record..... 17
III-C-1 - Relations entre Posts et Users..... 19
IV - 3ième jour: Gestion des utilisateurs..... 20
IV-A - Vue d'ensemble..... 20
IV-B - Authentification et Autorisation..... 22
IV-C - Création de la page 'LoginUser'..... 23
Création du gabarit..... 23
Création de la classe..... 24
Test..... 25
Ajout des liens de connexion/déconnexion à notre gabarit principal..... 25
IV-D - Création de la page nouvel utilisateur 'NewUser'..... 26
Création du gabarit..... 26
Création du fichier de classe..... 27
Test..... 28
Ajout de la vérification des droits d'accès..... 29
IV-E - Création de la page de mise à jour des utilisateurs 'EditUser'..... 30
Création du gabarit..... 30
Création du fichier de classe..... 32
Test..... 33
IV-F - Création de la page d'administration des utilisateurs 'AdminUser'..... 33
Création du gabarit..... 33
Création du fichier de classe..... 34
Ajout de la vérification des droits d'accès..... 35
Test..... 35
V - 4ième jour: Gestion des Messages..... 36
V-A - Vue d'ensemble de la gestion des messages..... 36
V-B - Création de la page d'affichage des messages 'ListPost'..... 38
Création du gabarit..... 38
Création du fichier de classe..... 39
Création du PostRenderer..... 40
Test..... 40
V-C - Création de la page détail d'un message 'ReadPost'..... 41
V-C-1 - Création du gabarit..... 41
Création du fichier de classe..... 42
Test..... 43
V-D - Création de la page nouveau message 'NewPost'..... 43
Création du gabarit..... 44

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création du fichier de classe........ 45
Test........ 45
V-E - Création de la page modification d'un message 'EditPost'........ 46
Création du gabarit........ 47
Création du fichier de classe........ 47
Test........ 48
VI - 5ième jour: Refactorisation et déploiement........ 49
VI-A - Utilisation des Thèmes et des Skins........ 49
Création des thèmes........ 50
Utilisation du thème........ 51
Test........ 51
VI-B - Gestion et journalisation d'erreur........ 52
Personnalisation de la gestion d'erreur........ 53
Journalisation des erreurs........ 54
Test........ 54
VI-C - Amélioration des performances........ 55
Changer le mode de fonctionnement de l'application........ 55
Activer le cache........ 55
Utilisation de pradolite.php........ 56
Autres techniques........ 56
VI-D - Résumé........ 56

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I - Introduction

I-A - Bienvenue dans ce tutoriel de création d'un Blog

Le but de ce tutoriel est de fournir aux nouveaux utilisateurs de PRADO, un guide pas à pas sur la façon de développer avec PRADO. Les lecteurs de ce tutoriel n'ont besoin d'aucune connaissance au sujet de PRADO. Par contre, les lecteurs doivent avoir des connaissances de base concernant la programmation orientée objet (POO) et les bases de données. Pour un guide plus exhaustif de PRADO, les lecteurs peuvent se référer au document **Quickstart Tutorial**.

Ce tutoriel est organisé sous la forme de journées. Chaque journée, des concepts et des techniques nouvelles de PRADO sont abordés et de nouvelles fonctionnalités sont mises en place. A la fin, nous obtiendrons un moteur de Blog simple mettant en place les fonctionnalités définies dans la section : buts à atteindre.

Pour pouvoir suivre ce tutoriel et créer les exemples pas à pas, les lecteurs doivent avoir accès à un serveur Web qui implémente PHP 5.1.0+ et PRADO 3.1+. Les instructions pour installer PRADO peuvent être trouvées ici : pradosoft.com.

I-B - Buts à atteindre

Cette page décrit les buts que nous allons nous fixer et développer avec PRADO. Nous n'allons pas mettre en place toutes les fonctionnalités d'un blog (ie : commentaires, organisation des messages, calendrier, etc.), parce que nous voulons garder ce tutoriel aussi court que possible et que nous espérons que ces fonctionnalités seront faciles à mettre en place après avoir fini ce tutoriel.

En général, les blog permettent aux utilisateurs de lire les messages et aux utilisateurs authentifiés de publier des messages. L'outil doit séparer la logique applicative de la couche présentation et il doit supporter le changement de thèmes graphiques.

Gestion des utilisateurs

- L'outil doit permettre de gérer les utilisateurs et leurs droits.
- L'outil doit permettre à l'administrateur de lister tous les utilisateurs.
- L'outil doit permettre à l'administrateur de créer des nouveaux utilisateurs.
- L'outil doit permettre à l'administrateur ou au propriétaire du compte de modifier son profil.
- L'outil doit permettre à l'administrateur de supprimer un utilisateur.

Gestion des messages

- L'outil doit permettre de lister les messages par ordre de date décroissante et d'en gérer la pagination.
- L'outil doit permettre de consulter le détail d'un message.
- L'outil doit permettre la création d'un nouveau message par un utilisateur authentifié.
- L'outil doit permettre la mise à jour d'un message par son auteur ou par l'administrateur.
- L'outil doit permettre la suppression d'un message par son auteur ou par l'administrateur.

Maintenance de l'outil

- L'outil doit permettre de collecter les retours utilisateurs.
- L'outil doit être assez flexible pour permettre l'intégration ultérieure de nouveaux composants.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- L'outil doit permettre de changer le thème qui régit la présentation des composants.

II - 1er jour : Découvrir PRADO

II-A - Installation

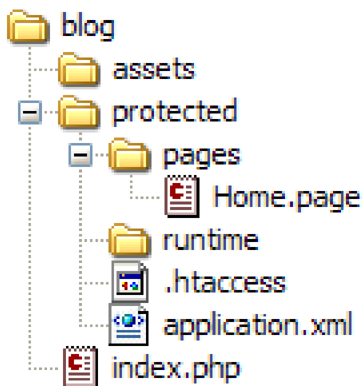
Nous commencerons par la mise en place de la structure des dossiers et fichiers requis par la plupart des applications développées avec PRADO. Nous allons utiliser **les outils en ligne de commande** pour atteindre ce but.

Nous partons du principe que le nom du dossier qui contiendra l'application est *blog* et que l'URL qui permet d'accéder à ce dossier est : *http://hostname/blog/* (remplacer *hostname* par le nom de votre serveur).

A l'intérieur du dossier blog, nous utilisons **les outils en ligne de commande** avec comme commande (remplacer *path/to* par le chemin d'installation du framework PRADO):

shell
```
php path/to/prado-cli.php -c .
```

L'utilisation de cette commande permet de créer la structure de dossiers et fichiers suivante :



Nous avons dorénavant, un squelette d'application PRADO accessible par l'URL http://hostname/blog/index.php et qui affiche une page contenant le message "Welcome to PRADO".

Il est de notre intérêt d'en apprendre plus à propos des dossiers et fichiers que nous venons de créer.

Les fichiers initiaux

Le script principal de l'application.

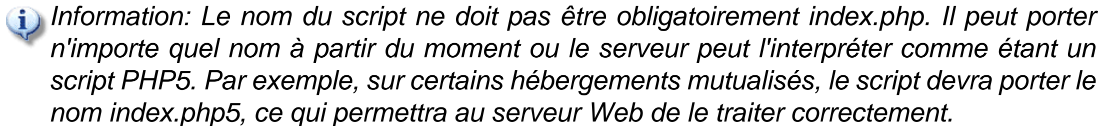
Toutes les applications PRADO ont un point d'entrée, habituellement nommé index.php. Dans la plupart des cas, c'est le seul script qui est directement accessible par les utilisateurs. Cela réduit les risques que les utilisateurs puissent lancer des scripts serveur auxquels ils ne devraient pas avoir accès.

Le but principal de ce script est d'initialiser l'environnement PRADO et de gérer toutes les requêtes utilisateurs. Ce script contient habituellement les commandes PHP suivantes,

php
```php
<?php
// include prado.php which contains basic PRADO classes
require_once('path/to/prado.php');
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

php
```php
// create a PRADO application instance
$application = new TApplication;
// run the application and handle user requests
$application->run();
?>
```



Information: Le nom du script ne doit pas être obligatoirement index.php. Il peut porter n'importe quel nom à partir du moment ou le serveur peut l'interpréter comme étant un script PHP5. Par exemple, sur certains hébergements mutualisés, le script devra porter le nom index.php5, ce qui permettra au serveur Web de le traiter correctement.

Le fichier de configuration de l'application

Le fichier optionnel XML *application.xml* contient la **configuration de l'application**. Son but principal est de permettre de configurer l'application qui sera créée par le script principal. Par exemple, nous pouvons activer le système de **log** pour notre application par le biais du fichier de configuration.

Le fichier *application.xml* est pour le moment presque vide. De ce fait, nous pouvons le supprimer parce que l'application n'utilise pour le moment que des fonctionnalités de base. Au fur et à mesure que nous avancerons, nous ferons référence régulièrement au fichier *application.xml* et vous expliquerons comment configurer l'application.

La page d'accueil

La page d'accueil *Home.page* (aussi dénommée page par défaut) est la seule **page** créée par les outils en ligne de commande de PRADO. C'est le contenu de ce fichier qui est affiché quand l'utilisateur navigue à l'adresse *http://hostname/blog/index.php*.

Le contenu du fichier *Home.page* respecte le **format de gabarit** qui pour la plupart du temps est du code HTML agrémenté de quelques balises spécifiques à PRADO. Par exemple, dans *Home.page* nous voyons du code HTML pur :

html
```html
<html>
<head>
  <title>Welcome to PRADO</title>
</head>
<body>
<h1>Welcome to PRADO!</h1>
</body>
</html>
```

Les dossiers initiaux

Le dossier *protected*

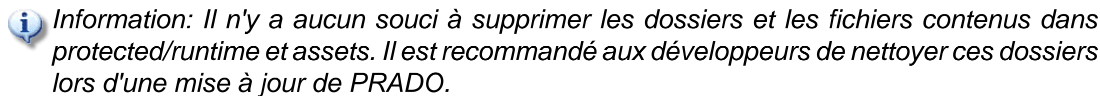
Le dossier *protected*, aussi connu sous le nom chemin de base de l'application, est le dossier racine qui contient les pages, les gabarits, les fichiers de configuration, les données, etc. Le nom *protected* indique que ce dossier doit être masqué des personnes qui consultent le site, ceci parce que les fichiers dans ce dossier contiennent la plupart du temps des données sensibles.

Les différents serveurs Web ont différents moyens de "protéger" un dossier. Pour Apache, le moyen le plus simple est de créer dans le dossier un fichier nommé .htaccess avec le contenu *deny from all*.

Les dossiers *protected/runtime* et *assets*

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Les dossiers *protected/runtime* et *assets* sont deux dossiers qui doivent avoir l'autorisation "en écriture" pour le serveur Web. Le dossier *runtime* contient des données sensibles (ie: fichier de configuration déjà analysé) générées à l'exécution de PRADO tandis que le dossier *assets* contient les ressources qui doivent être publiques (ie: les images, les fichiers javascript).



Information: Il n'y a aucun souci à supprimer les dossiers et les fichiers contenus dans protected/runtime et assets. Il est recommandé aux développeurs de nettoyer ces dossiers lors d'une mise à jour de PRADO.

Le dossier pages

Le dossier *pages* est le *dossier racine* qui contient toutes les **pages** d'une application PRADO. Il est l'équivalent du dossier *htdocs* d'un serveur Web Apache.

Nous avons déjà vu comment accéder à la page d'accueil. Pour accéder à n'importe quelle page situé dans le dossier *pages*, il faut utiliser l'URL suivante *http://hostname/blog/index.php?page=chemin.vers.NomdelaPage*. En fonction de cette URL, PRADO recherche une page dénommée *NomdelaPage* dans le dossier *pages/chemin/vers*. L'URL que nous avons utilisée précédemment pour accéder à la page d'accueil correspond à *http://hostname/blog/index.php?page=Home*.

Personnalisation

Il est tout à fait possible de personnaliser le nom et l'emplacement des fichiers et dossiers décrit précédemment.

Par exemple, pour améliorer la sécurité, certains pourraient désirer déplacer la totalité du dossier *protected* à un emplacement inaccessible par le Web. Pour faire cela, utilisez la commande PHP suivante pour initialiser l'instance de l'application PRADO dans le script principal :

```php
$application = new TApplication( 'path/to/protected' );
```

Pour changer l'emplacement du dossier racine des pages et le nom de la page d'accueil, il est possible de modifier le fichier de configuration *application.xml* de cette manière :

```xml
<?xml version="1.0" encoding="utf-8"?>
<application id="blog" mode="Debug">
  <services>
    <service id="page"
           class="TPageService"
           BasePath="path.to.pages"
           DefaultPage="NewHome"
           />
  </services>
</application>
```

En avançant dans l'apprentissage de PRADO, vous verrez que PRADO est très souple et qu'il est possible de personnaliser la plupart des comportements de base. Nous décrirons d'autres techniques au fur et à mesure de ce tutoriel.

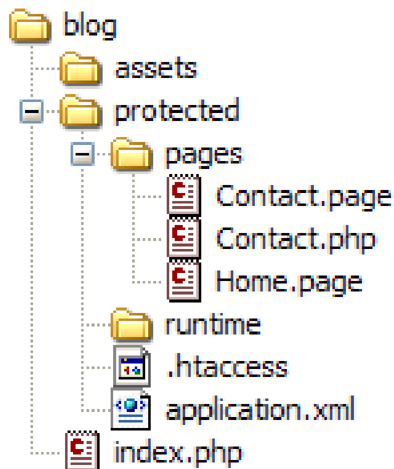
II-B - Création de la page Contact

Nous avons créé une page par défaut *Home.page* en utilisant les **outils en ligne de commande de PRADO**. Cette page est relativement statique parce qu'elle ne contient que du contenu HTML. Dans cette session, nous allons créer une page dynamique dénommée *Contact*.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Le but de cette page est de collecter les retours d'informations des utilisateurs Web concernant notre outil de blog. Pour atteindre ce but, nous envisageons d'utiliser un formulaire qui sera à remplir. Dans ce formulaire, nous demanderons le nom de l'utilisateur, son adresse email et son commentaire. Après que le formulaire ai été rempli et envoyé, un email avec le commentaire sera envoyé à l'administrateur.

Pour créer la page *Contact*, nous avons besoin de 2 fichiers dans le dossier *pages* : le fichier de gabarit *Contact.page* et le fichier de classe PHP *Contact.php*.



Information: Une ***page*** *doit avoir un fichier de* ***gabarit*** *(extension .page) ou un fichier de classe PHP, ou les deux :*
- Une page avec seulement un gabarit est généralement une page statique, comme la page d'accueil que nous avons déjà créée.
- Une page avec seulement un fichier de classe PHP produit le code HTML directement à partir de l'exécution du script.
- Une page avec un gabarit et un fichier de classe PHP combine les avantages des deux : un gabarit pour facilement organiser la présentation de la page et un fichier de classe PHP pour produire le contenu dynamique.

Création de la page gabarit

Nous allons premièrement créer le fichier gabarit de la page *Contact*.

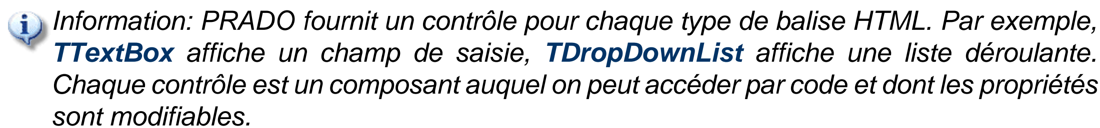
Nous utilisons un fichier gabarit pour organiser la présentation de notre formulaire. Dans notre gabarit, nous utilisons des **champs de saisie** pour collecter le nom de l'utilisateur, son email et son commentaire. D'autre part, nous utilisons des **validateurs** pour nous assurer que l'utilisateur a bien fourni les éléments avant d'envoyer le formulaire. Le contenu complet du gabarit est le suivant :

xml
```xml
<html>
<head><title>Mon Blog - Contact</title></head>
<body>
<h1>Contact</h1>
<p>Veuillez remplir le formulaire suivant pour me laisser vos impressions au sujet de mon blog.
 Merci !</p>

<com:TForm>

<span>Votre nom:</span>
<com:TRequiredFieldValidator ControlToValidate="Name"
    ErrorMessage="Veuillez indiquer votre nom."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Name" />
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

xml
```xml
<br/>
<span>Votre email:</span>
<com:TRequiredFieldValidator ControlToValidate="Email"
    ErrorMessage="Veuillez indiquer votre email."
    Display="Dynamic" />
<com:TEmailAddressValidator ControlToValidate="Email"
    ErrorMessage="Vous avez saisi un email invalide."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Email" />

<br/>
<span>Commentaires:</span>
<com:TRequiredFieldValidator ControlToValidate="Feedback"
    ErrorMessage="Veuillez saisir un commentaire."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Feedback"
    TextMode="MultiLine"
    Rows="10"
    Columns="40" />

<br/>
<com:TButton Text="Envoyer" OnClick="submitButtonClicked" />

</com:TForm>

</body>
</html>
```

Comme vous pouvez le voir, un fichier gabarit ressemble énormément à un fichier HTML classique. La principale différence concerne le fichier gabarit qui contient quelques balises *<com:>*. Chaque balise *<com:>* fait référence à un **contrôle** dont les propriétés sont initialisées grâce aux paires nom-valeur de la balise. Par exemple, la balise *<com:TButton>* fait référence au contrôle **TButton** qui affiche un bouton permettant à l'utilisateur de soumettre le formulaire. Pour une syntaxe complète, veuillez vous référer au **Tutoriel de démarrage rapide**.



Information: PRADO fournit un contrôle pour chaque type de balise HTML. Par exemple, ***TTextBox*** *affiche un champ de saisie,* ***TDropDownList*** *affiche une liste déroulante. Chaque contrôle est un composant auquel on peut accéder par code et dont les propriétés sont modifiables.*

Avant le contrôle *TTextBox*, le gabarit utilise aussi plusieurs validateurs qui permettent de s'assurer que les données saisies sont bien conformes à notre attente. Par exemple, pour nous assurer que l'adresse email est valide, nous utilisons les deux validateurs suivants,

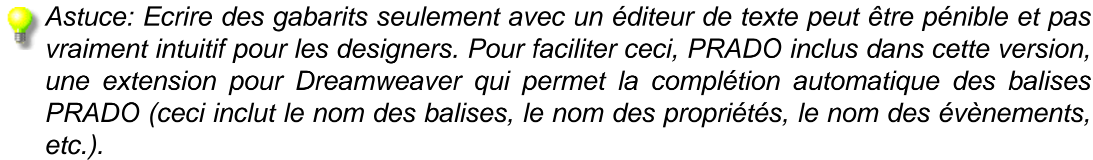
xml
```xml
<span>Your Email:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Email"
    ErrorMessage="Veuillez indiquer votre email."
    Display="Dynamic" />
<com:TEmailAddressValidator
    ControlToValidate="Email"
    ErrorMessage="Vous avez saisi un email invalide."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Email" />
<br/>
```

Ci-dessous, un résumé des contrôles utilisés dans le gabarit :

- **TForm** affiche un formulaire HTML. Chaque contrôle de saisie doit être au sein d'un TForm. Et le plus important, au plus un TForm peut apparaitre dans une page.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- **TTextBox** affiche un champ de saisie utilisateur.
- **TRequiredFieldValidator** s'assure que le contrôle de saisie associé n'est pas vide quand le formulaire est envoyé.
- **TEmailAddressValidator** s'assure que le champ de saisie contient une adresse email valide quand le formulaire est envoyé.



Astuce: Ecrire des gabarits seulement avec un éditeur de texte peut être pénible et pas vraiment intuitif pour les designers. Pour faciliter ceci, PRADO inclus dans cette version, une extension pour Dreamweaver qui permet la complétion automatique des balises PRADO (ceci inclut le nom des balises, le nom des propriétés, le nom des évènements, etc.).

Création du fichier de classe PHP

Nous allons maintenant créer le fichier de classe PHP *Contact.php*. Ce fichier est nécessaire parce que nous devons agir après la soumission du formulaire.

Notez les lignes dans le fichier gabarit. Elles indiquent que lorsque l'utilisateur soumet le formulaire, la méthode *submitButtonClicked()* doit être appelé. Ici, *OnClick* est le nom de l'évènement et la méthode correspondante doit être défini dans le fichier de classe PHP.

xml
```xml
<com:TButton Text="Submit" OnClick="submitButtonClicked" />
```

Nous écrirons donc le fichier de classe suivant :

php
```php
<?PHP
class Contact extends TPage
{
    /**
     * Gestionnaire d'évènement pour OnClick (bouton submit).
     * @param TButton le bouton qui a générer l'évènement
     * @param TEventParameter les paramètres de l'évènement (null dans ce cas)
     */
    public function submitButtonClicked($sender, $param)
    {
        if ($this->IsValid)  // vérifie que les validations sont Ok
        {
            // récupère le nom de l'utilisateur, son email et son commentaire
            $name = $this->Name->Text;
            $email = $this->Email->Text;
            $feedback = $this->Feedback->Text;

            // envoie un email à l'administrateur avec les informations
            $this->mailFeedback($name, $email, $feedback);
        }
    }

    protected function mailFeedback($name, $email, $feedback)
    {
        // implémentation de l'envoi de l'email
    }
}
?>
```

Le code précédent est largement explicite. En fait, nous avons juste montré le principe d'un gestionnaire d'évènement. Dans le gestionnaire d'évènement *submitButtonClicked()*, nous récupérons les éléments saisies par l'utilisateur. Par exemple, *$this->Name->Text* retourne la valeur de la propriété *Text* du contrôle *Name* qui est un contrôle permettant la saisie du nom de l'utilisateur.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Information: Le nom de la classe héritant de TPage doit être le même que le nom du fichier. C'est aussi une nécessité pour écrire n'importe quelle classe de composant PRADO.

Test

Notre nouvelle page *Contact* peut être testée en naviguant à l'URL *http://hostname/blog/index.PHP?page=Contact*. Si vous cliquez sur le bouton "envoyer" sans avoir saisi de données, vous verrez apparaitre des messages d'erreurs à côté des champs de saisie. Si vous entrez toutes les informations nécessaires, la méthode *mailFeedback()* sera appelée.

Contact

Please fill out the following form to let me know your feedback on my blog. Thanks!

Your Name:

Your Email:

Feedback:

Submit

Contact

Please fill out the follo
feedback on my blog.

Your Name: Please p

Your Email: Please pr

Feedback: Please pro

Submit

Une amélioration possible à notre page serait d'afficher un message de confirmation après que l'utilisateur ai envoyé le formulaire. Il serait aussi envisageable de rediriger le navigateur vers une adresse différente si toutes les informations ont été saisies correctement. Nous laisserons aux lecteurs la mise en place de ces fonctionnalités.

Information: Chaque validateur représente une règle de validation. Un champ de saisie unique peut être associé à un ou plusieurs validateurs. Les validateurs effectuent les vérifications aussi bien du côté client que du côté serveur. Côté client (navigateur), les validations sont effectuées grâce à du javascript, côté serveur, elles sont effectuées en PHP. Les validations côté client peuvent être désactivées tandis que celles côté serveur ne peuvent l'être. Ceci permet de s'assurer que les règles de validation sont toujours appliquées.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

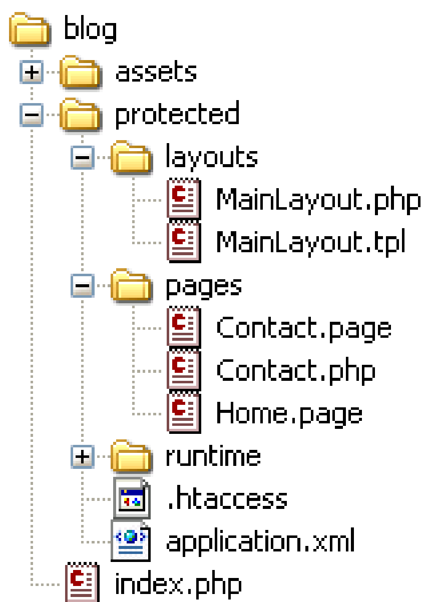
II-C - Partager les modèles de gabarit

Dans cette section, nous allons utiliser la fonctionnalité **gabarit principal/contenu** de PRADO pour partager une mise en page commune sur tout notre site. Les mises en page communes font référence aux parties qui sont identiques ou presque pour un ensemble de pages. Par exemple, dans notre outil de blog, toutes les pages partagent le même entête, pied de page et la même barre latérale contenant les liens. La solution la plus radicale est de répéter sur chaque page les parties communes. Par contre, cette approche est une source d'erreurs et difficile à maintenir. La fonctionnalité **gabarit principal/contenu** nous permets de traiter les parties communes comme un contrôle qui centralise la logique applicative et la présentation de chaque page.

*Information: Il est aussi possible de partager les parties communes grâce à l'**inclusion de gabarits**, un peu comme l'inclusion de fichier php. L'inconvénient de l'inclusion de gabarits est que l'on ne peut pas partager la logique applicative.*

Création du gabarit principal

Nous allons maintenant créer le gabarit principal *MainLayout* qui représente les parties communes partagées par toutes nos pages. Le contrôle *MainLayout* est un contrôle de gabarit qui hérite de *TTemplateControl*. Il a besoin d'un fichier de gabarit *MainLayout.tpl* et d'un fichier de classe *MainLayout.php* situés dans le même dossier. Pour faciliter la maintenance, nous allons créer le nouveau dossier *protected/layouts* pour les accueillir.



Pour le moment, *MainLayout* contient seulement un entête simple et un pied de page, comme décrit ci-après. Plus tard, nous ajouterons une barre latérale. Les lecteurs sont encouragés à ajouter des fonctionnalités.

xml
```xml
<html>
<com:THead />
<body>
<com:TForm>
<div id="page">

<div id="header">
<h1>Mon blog PRADO</h1>
</div>

<div id="main">
<com:TContentPlaceHolder ID="Main" />
</div>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

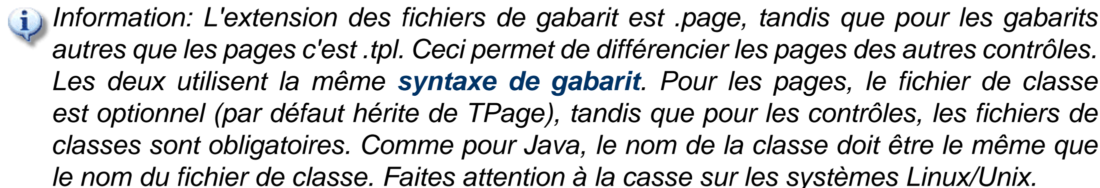
```xml
<div id="footer">
<%= PRADO::poweredByPrado() %>
</div>

</div>
</com:TForm>
</body>
</html>
```

Ci-dessus, le contenu du fichier de gabarit *MainLayout.tpl*. Trois nouvelles balises sont utilisées.

- <com:TContentPlaceHolder> represente un contrôle TContentPlaceHolder. Il définit l'emplacement dans le gabarit où le contenu sera inséré. Dans notre cas, le contenu proviendra des pages qui utiliseront notre gabarit principal.
- <com:THead> représente un contrôle THead qui correspond à la balise <head> d'un document HTML. Ceci permet à PRADO de manipuler la balise <head> comme un composant (ie: définir le titre de la page, ajouter des feuilles de styles CSS.)
- <%= %> est une balise de configuration. Elle affiche le résultat de l'évaluation de l'expression à l'endroit où elle se situe.

Le fichier de classe MainLayout.php est très simple :

```php
<?php
class MainLayout extends TTemplateControl
{
}
?>
```



Information: L'extension des fichiers de gabarit est .page, tandis que pour les gabarits autres que les pages c'est .tpl. Ceci permet de différencier les pages des autres contrôles. Les deux utilisent la même ***syntaxe de gabarit****. Pour les pages, le fichier de classe est optionnel (par défaut hérite de TPage), tandis que pour les contrôles, les fichiers de classes sont obligatoires. Comme pour Java, le nom de la classe doit être le même que le nom du fichier de classe. Faites attention à la casse sur les systèmes Linux/Unix.*

Utilisation du gabarit principal

Pour utiliser notre gabarit principal nouvellement créé, nous allons modifier nos fichiers *Home.page* et *Contact.page*. En particulier, nous devons supprimer les entêtes et pied de page parce que le gabarit principal a la responsabilité de les afficher ; par ailleurs, nous devons indiquer aux deux pages que leur gabarit principal est *MainLayout*.

Ci-dessous, le contenu de *Contact.page* après les modifications :

```xml
<%@ MasterClass="Application.layouts.MainLayout" Title="Mon blog - Contact" %>

<com:TContent ID="Main">

<h1>Contact</h1>
<p>Veuillez remplir le formulaire suivant pour me laisser vos impressions au sujet de mon blog.
 Merci !</p>

...champs de saisie et validateurs pour le nom d'utilisateur...

...champs de saisie et validateurs pour l'email...

...champs de saisie et validateurs pour le commentaire...
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

xml
```xml
<com:TButton Text="Envoyer" OnClick="submitButtonClicked" />

</com:TContent>
```

Le contenu entre les balises *<com:TContent>* sera inséré dans l'emplacement réservé par *<com:TContentPlaceHolder>* dans le gabarit principal.

Information: Il est possible d'avoir plusieurs TContentPlaceHolder dans un gabarit principal et plusieurs TContent dans un fichier de contenu. Ils sont associés par leurs propriétés ID. Il est aussi possible de définir un contenu comme étant le gabarit principal d'un autre contenu, ceci en plaçant une balise TContentPlaceHolder à l'endroit désiré. Ceci est appelé gabarits principaux imbriqués

A côté de la balise *<com:TContent>*, nous avons vu une nouvelle balise *<%@ %>*, qui est dénommé une **balise de contrôle de gabarit**. Elle contient des paires nom-valeur utilisées pour initialiser les propriétés correspondantes du propriétaire de gabarit, dans notre cas, la page *Contact*.

En définissant la propriété *MasterClass* comme étant de type *Application.layouts.MainLayout*, nous avons indiqué à la page *Contact* d'utiliser *MainLayout* comme gabarit principal. Ici, nous avons utilisé un espace de noms pour nous référer à la classe *MainLayout*.

Information: Les espaces de noms sont largement utilisés en programmation PRADO. Ils sont utilisés conjointement avec les alias de chemins. PRADO définit deux alias de chemins: System fait référence au dossier framework de l'installation PRADO, et Application fait référence au dossier protected. L'espace de noms Application.layouts.MainLayout peut ainsi être traduit par protected/layouts/MainLayout ce qui est précisément le nom du fichier (sans l'extension .php) de la classe MainLayout.

Autres possibilités pour spécifier le gabarit principal

Il y a plusieurs alternatives pour spécifier le gabarit principal.

Vous pouvez définir le gabarit principal comme ci-dessous pour pouvoir en changer dynamiquement.

php
```php
<?php
class Contact extends TPage
{
    public function onPreInit($param)
    {
        parent::onPreInit($param);
        $this->MasterClass='Path.To.NewLayout';
    }

    // ...
}
?>
```

Ci-dessus, nous indiquons d'utiliser le gabarit principal *MasterClass* dans la méthode *onPreInit()* qui est héritée de *TPage*. Cette méthode est appelé par PRADO juste après que l'instance de la page soit créée. Nous pouvons ainsi déclarer au moment où la page est requise quel gabarit principal utiliser. Par exemple, quand la page est requise par un utilisateur enregistré, nous pouvons utiliser le gabarit A, et le gabarit B si l'utilisateur qui demande la page est un invité.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Nous pouvons aussi spécifier quel gabarit principal utiliser dans le fichier de **configuration de l'application** ou encore dans le fichier de **configuration de la page**. Ci-dessous, le fichier de configuration de l'application modifié pour notre blog.

xml
```xml
<?xml version="1.0" encoding="utf-8"?>
<application id="blog" mode="Debug">
  <!-- configuration for available services -->
  <services>
    <service id="page" class="TPageService" DefaultPage="Home">
      <!-- initial properties set for all pages -->
      <pages MasterClass="Application.layouts.MainLayout" />
    </service>
  </services>
</application>
```

En faisant cela, nous évitons de définir le gabarit principal dans chaque page. Si nous décidons d'utiliser un autre gabarit principal, il nous suffit de changer le fichier de configuration de l'application. Pour cette raison, dans notre blog, nous utiliserons cette approche.

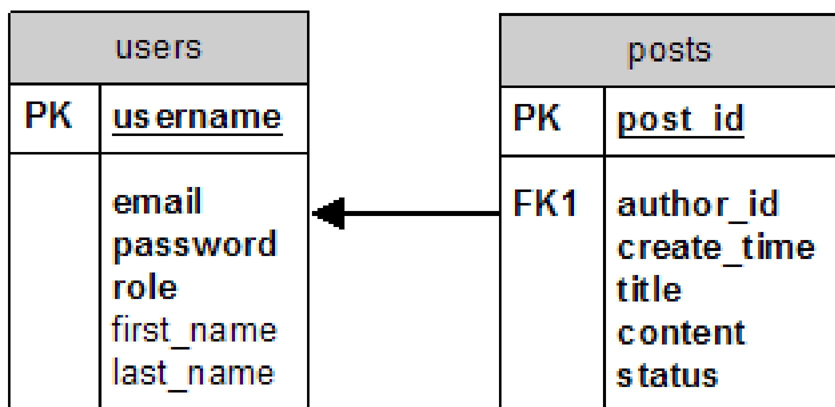
Information: Il y a un ordre qui permet de savoir quel fichier gabarit principal utiliser s'il est spécifié à plusieurs endroits. En particulier onPreInit() est prioritaire au fichier de configuration de la page qui est lui même prioritaire au fichier de configuration de l'application. Ainsi, si vous spécifiez MainLayout dans le fichier de configuration de l'application/page et que vous spécifiez SpecialLayout dans Contact.page, ce sera le dernier qui sera pris en compte.

III - 2ième jour: Mise en place de la Base de Données

III-A - Création de la base

La plupart des applications Web utilisent une base de données pour conserver les informations. Notre blog n'est pas une exception. Dans cette section, nous allons décrire comment écrire une application qui interagit avec une base de données. Nous allons utiliser les deux moyens suivants **database access object (DAO)** et **Active Record**.

Pour ce tutoriel, nous avons simplifié les besoins, nous aurons juste à gérer les utilisateurs et les messages. Nous allons donc créer deux tables *users* et *posts*, comme décrit dans le diagramme ci-après.



Nous utilisons une base de données SQLite 3 pour conserver nos données. La première étape est de convertir notre diagramme en commandes SQL et de l'enregistrer dans le fichier *protected/schema.sql*.

```sql
/* création de la table utilisateurs */
CREATE TABLE users (
  username      VARCHAR(128) NOT NULL PRIMARY KEY,
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

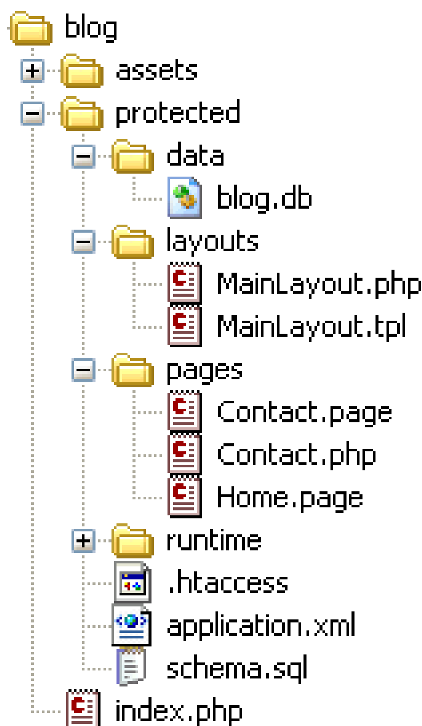
```
  email          VARCHAR(128) NOT NULL,
  password       VARCHAR(128) NOT NULL,    /* mot de passe en clair */
  role           INTEGER NOT NULL,         /* 0: utilisateur normal, 1: administrateur */
  first_name     VARCHAR(128),
  last_name      VARCHAR(128)
);

/* création de la table messages */
CREATE TABLE posts (
  post_id        INTEGER NOT NULL PRIMARY KEY,
  author_id      VARCHAR(128) NOT NULL
                 CONSTRAINT fk_author REFERENCES users(username),
  create_time    INTEGER NOT NULL,         /* UNIX timestamp */
  title          VARCHAR(256) NOT NULL,    /* titre du message */
  content        TEXT,                     /* corps du message  */
  status         INTEGER NOT NULL          /* 0: publié; 1: brouillon; 2: en attente; 2: accès interdit */
);

/* insertion de quelques données initiales */
INSERT INTO users VALUES ('admin', 'admin@example.com', 'demo', 1, 'Qiang', 'Xue');
INSERT INTO users VALUES ('demo', 'demo@example.com', 'demo', 0, 'Wei', 'Zhuo');
INSERT INTO posts VALUES (NULL, 'admin', 1175708482, 'first post', 'this is my first post', 0);
```

*Note: La contrainte fk_author est ignorée par SQlite vu que SQLite ne supporte pas les **clés étrangères**. Néanmoins, nous gardons cette contrainte dans le but de pouvoir porter notre blog sur d'autres SGBD. Par ailleurs, nous utilisons la capacité **auto incrémenté** du champ posts.post_id si celui-ci est renseigné à NULL lors de l'insertion.*

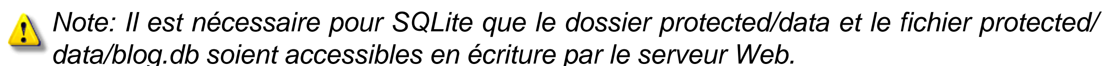
Nous utilisons après ceci, les **outils en ligne de commande SQLite** pour créer la base de données. Nous avons créé un dossier *protected/data* qui contiendra le fichier de base de données. Nous lançons maintenant la ligne de commande suivante dans le dossier *protected/data*.

```
sqlite3 blog.db < ../schema.sql
```

La base de données est ainsi créée dans le fichier *protected/data/blog.db* et nous pouvons maintenant voir la l'arborescence suivante :



Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

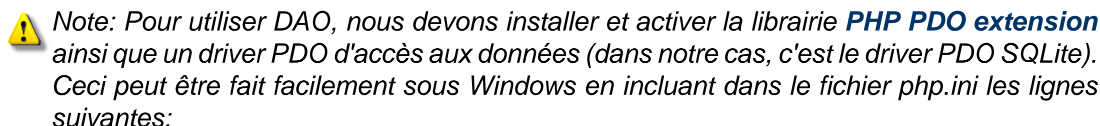


Note: Il est nécessaire pour SQLite que le dossier protected/data et le fichier protected/data/blog.db soient accessibles en écriture par le serveur Web.

III-B - Connexion à la base

Pour pouvoir utiliser la base de données que nous venons de créer, nous devons tout d'abord établir une connexion.

Nous allons utiliser **Data Access Objects (DAO)** pour établir une couche d'abstraction. Si dans le futur nous décidions d'utiliser un autre SGBD (ie: PostgreSQl, Oracle, ...), il nous suffirait de modifier la chaine de connexion (DSN) et nous pourrions conserver notre code PHP intact.



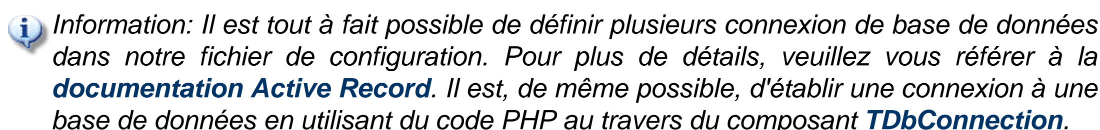
Note: Pour utiliser DAO, nous devons installer et activer la librairie ***PHP PDO extension*** *ainsi que un driver PDO d'accès aux données (dans notre cas, c'est le driver PDO SQLite). Ceci peut être fait facilement sous Windows en incluant dans le fichier php.ini les lignes suivantes:*
** extension=php_pdo.dll*
** extension=php_pdo_sqlite.dll*

Pour une meilleure abstraction de notre base de données, nous utilisons aussi la fonctionnalité **Active Record (AR)**. Chaque enregistrement est ainsi représenté par un objet qui a la possibilité d'exécuter des requêtes, de mettre à jour les données, de les supprimer et ceci sans écrire la moindre commande SQL.

Nous modifions notre fichier de configuration de l'application *protected/application.xml* en insérant les lignes suivantes, qui indiquent à *Active Record* comment se connecter à notre base de données.

```
<modules>
  <module id="db" class="System.Data.TDataSourceConfig">
    <database ConnectionString="sqlite:protected/data/blog.db" />
  </module>
  <module class="System.Data.ActiveRecord.TActiveRecordConfig" ConnectionID="db" />
</modules>
```

Dans la configuration précédente, nous avons ajouté deux **modules** à notre application. Le module *TDataSourceConfig* est configuré avec la chaine de connexion *sqlite:protected/data/blog.db* qui pointe vers notre base de données. Cette connexion est utilisée par le module *TActiveRecordConfig* qui est requis pour l'utilisation d'*Active Record*.



Information: Il est tout à fait possible de définir plusieurs connexion de base de données dans notre fichier de configuration. Pour plus de détails, veuillez vous référer à la ***documentation Active Record****. Il est, de même possible, d'établir une connexion à une base de données en utilisant du code PHP au travers du composant* ***TDbConnection****.*

III-C - Création des classes Active Record

Nous avons besoin de définir deux classes **Active Record**, *UserRecord* et *PostRecord*, qui représentent respectivement les tables *users* et *posts*. Les classes *Active Record* doivent hériter de la classe ActiveRecord, et doivent définir les propriétés qui correspondent aux champs de la table correspondante.

Pour une meilleure organisation de notre arborescence, nous créons un nouveau dossier *protected/database* qui contiendra nos deux classes. Nous modifions notre fichier de configuration de l'application en y insérant les lignes suivantes. Ceci est équivalent à inclure le dossier *protected/database* à notre chemin d'inclusion de PHP (*include_path*). Cela nous permet d'utiliser nos classes sans avoir besoin de les inclure explicitement.

```
<paths>
  <using namespace="Application.database.*" />
</paths>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Au lieu de créer nos classes manuellement, nous allons utiliser les **outils en ligne de commande de PRADO** pour qu'il nous génère les classes.

Dans le dossier *blog*, lancer la commande suivante pour entrer dans le mode interactif de l'outil en ligne de commande :

```
php path/to/prado-cli.php shell .
```

Vous devriez voir :

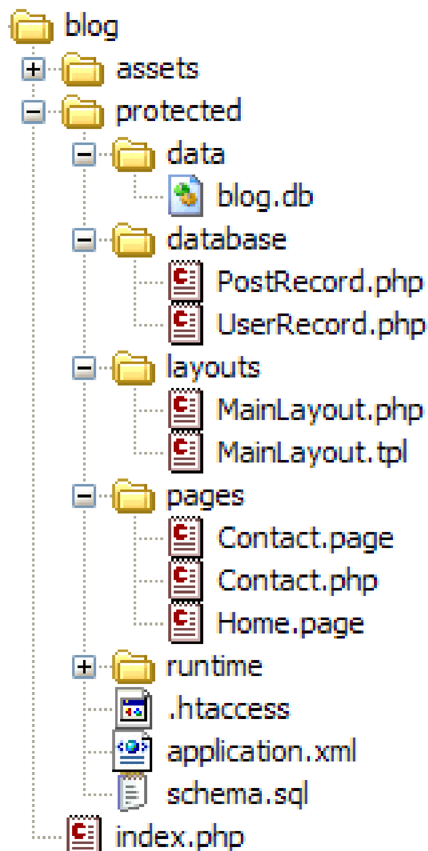
```
Command line tools for Prado 3.1.0.
** Loaded PRADO appplication in directory "protected".
PHP-Shell - Version 0.3.1
(c) 2006, Jan Kneschke <jan@kneschke.de>

>> use '?' to open the inline help

>>
```

A l'invite de commande, entrer les deux commandes suivantes pour créer les classes *UserRecord* et *PostRecord*.

```
>> generate users Application.database.UserRecord

>> generate posts Application.database.PostRecord
```

Ici, nous avons utilisés les **espaces de noms** pour indiquer où les classes seront créées. Le chemin *Application.database.UserRecord* indique que nous désirons que la classe soit insérée dans *protected/database/UserRecord.php*.

Nous devrions voir l'arborescence suivante ainsi que nos deux nouveaux fichiers dans le dossier *protected/database* :



Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Si vous consultez le fichier *PostRecord*, vous devriez voir le contenu suivant :

```
class PostRecord extends TActiveRecord
{
    const TABLE='posts';
    public $post_id;
    public $author_id;
    public $create_time;
    public $title;
    public $content;
    public $status;

    public static function finder($className=__CLASS__)
    {
        return parent::finder($className);
    }
}
```

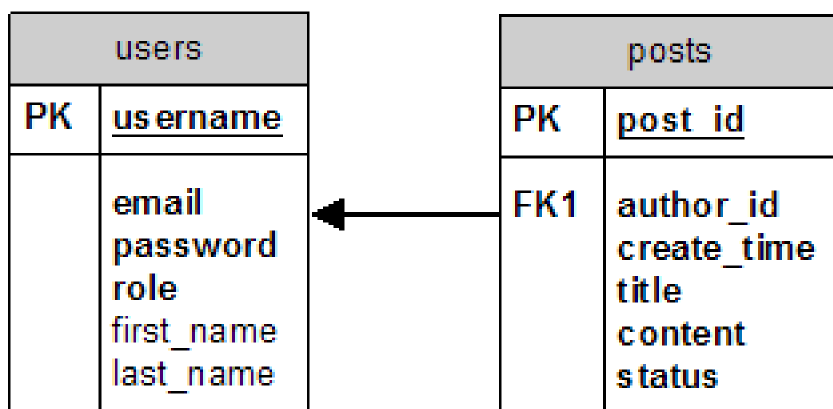
Comme vous pouvez le constater, pour chaque champs de la table *posts*, la classe déclare un membre correspondant. La constante *TABLE* indique le nom de la table que gère la classe *PostRecord*. La méthode statique *finder()* permet d'effectuer des requêtes et de lire les données sous forme d'objets *PostRecord*.

Vous pouvez utiliser l'outil en ligne de commande pour tester nos nouvelles classes. En restant dans le mode interactif de l'outil en ligne de commande, vous pouvez saisir les commandes PHP et voir ce qui suit. Vous pouvez tester des commandes telles que *UserRecord::finder()->findAll()*.

```
>> PostRecord::finder()->findAll()
array
(
    [0] => PostRecord#1
    (
        [post_id] => '1'
        [author_id] => 'admin'
        [create_time] => '1175708482'
        [title] => 'first post'
        [content] => 'this is my first post'
        [status] => '0'
        [TActiveRecord:_readOnly] => false
        [TActiveRecord:_connection] => null
        [TComponent:_e] => array()
    )
)
```

III-C-1 - Relations entre Posts et Users

Rappellez-vous qu'il y a une relation entre les tables *users* et *posts*. Le diagramme des relations est indiqué ci-après.



Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

A partir de ce diagramme, nous voyons que la table *posts* contient un champ *author_id*. Le champ *author_id* est une clé étrangère qui référence la table *users*. En particulier, les valeurs du champ *author_id* doivent apparaitre dans dans le champ *username* de la table *users*. Une des conséquences de cette relation, en réfléchissant orienté objet, est que chaque "post" repose sur un "author" et qu'un "author" peut avoir plusieurs "posts".

Nous pouvons modéliser les relations entre *posts* and *users* dans nos classes *PostRecord* and *UserRecord* en les modifiant comme ci-dessous :

```
class PostRecord extends TActiveRecord
{
    //... propriétés et méthodes comme précédemment

    public $author; //contient un objet UserRecord

    protected static $RELATIONS=array
    (
        'author' => array(self::BELONGS_TO, 'UserRecord'),
    );
}
```

La propriété statique *$RELATIONS* de la classe *PostRecord* définit que la propriété *$author* fait référence à un *UserRecord*. Dans le tableau : *array(self::BELONGS_TO, 'UserRecord')*, le premier élément défini le type de relation, dans notre cas, *self::BELONGS_TO*. Le deuxième élément est le nom de l'objet en relation, dans notre cas *UserRecord*. La classe *UserRecord* est définie comme ci-dessous, la différence est que chaque objet *UserRecord* contient plusieurs *PostRecord*.

```
class UserRecord extends TActiveRecord
{
    //... propriétés et méthodes comme précédemment

    public $posts=array(); //contient un tableau de PostRecord

    protected static $RELATIONS=array
    (
        'posts' => array(self::HAS_MANY, 'PostRecord'),
    );
}
```

Un tableau de *UserRecord* ainsi que les messages correspondants peuvent être lu de la manière suivante :

```
$users = UserRecord::finder()->withPosts()->findAll();
```

Astuce: La méthode withXXX() (avec XXX qui est le nom de la propriété de la relation, dans notre cas Posts) lit les données correspondantes de PostRecords en utilisant une deuxième requête (mais pas en utilisant une jointure). La méthode withXXX() accepte les mêmes arguments que les autres méthodes finder de l'objet Active record, ie : withPosts('status = ?', 0).

Plus d'informations sont disponibles dans le manuel de de démarrage rapide **Active Record**.

IV - 3ième jour: Gestion des utilisateurs

IV-A - Vue d'ensemble

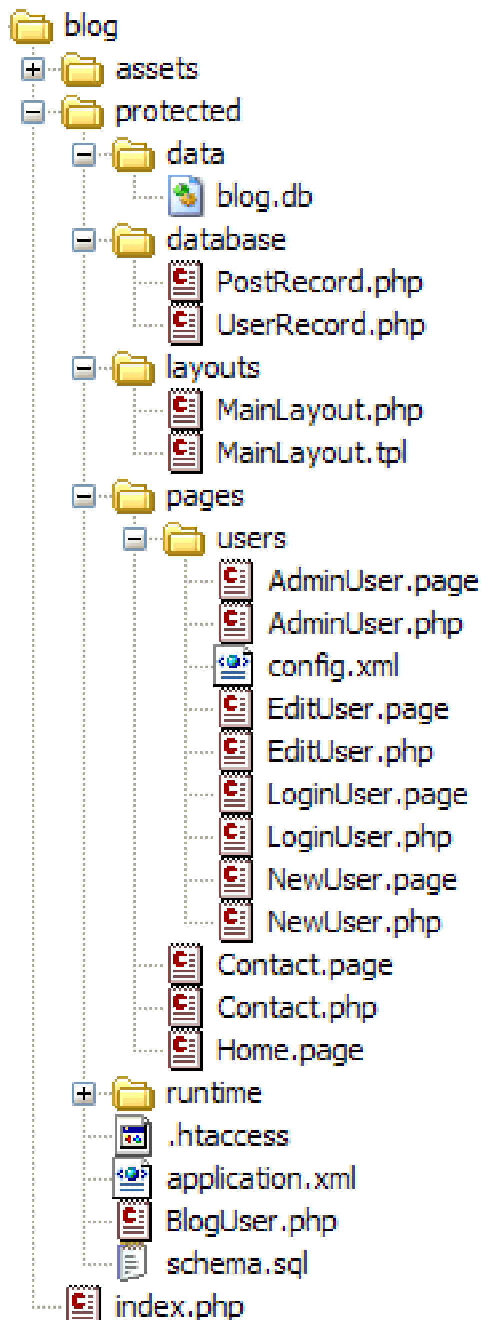
Dans cette section, nous allons créer les pages concernant la gestion des utilisateurs. En particulier, nous allons implémenter les fonctionnalités suivantes: connexion/déconnexion des utilisateurs, création d'un utilisateur, mise à jour/suppression des utilisateurs.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

En accord avec les buts à atteindre, nous devons créer les pages suivantes. Pour une meilleure organisation de notre code, ces pages seront créées dans le dossier *protected/pages/users*.

- *LoginUser* affiche le formulaire de connexion.
- *NewUser* pour la création d'un compte utilisateur.
- *EditUser* permet à un utilisateur enregistré de mettre à jour son profil.
- *AdminUser* permet à l'administrateur de gérer les comptes utilisateurs, y compris la gestion des droits d'accès et la suppression d'un compte.

Après avoir fini cette section, nous devrions obtenir l'arborescence suivante :



Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

IV-B - Authentification et Autorisation

Avant que nous n'implémentions la gestion des utilisateurs, nous devons activer les modules **authentification et autorisation**.

Nous ajoutons deux nouveaux modules à notre fichier de configuration de l'application.

```
<modules>
    ...modules TDataSourceConfig et TActiveRecordConfig ...

    <module id="auth"
      class="System.Security.TAuthManager"
      UserManager="users"
      LoginPage="users.LoginUser" />

    <module id="users"
      class="System.Security.TDbUserManager"
      UserClass="Application.BlogUser" />
</modules>
```

Le module **TAuthManager** gère le processus d'authentification et d'autorisation. Il utilise le module *users* comme gestionnaire d'utilisateur (voir ci-après). En spécifiant la propriété *LoginPage*, nous indiquons au module d'authentification de rediriger vers la page *LoginUser* quand il est nécessaire de s'authentifier. Nous décrirons comment créer la page *LoginUser* ci-après.

Le module *user* est une classe de type **TDbUserManager** qui est responsable de la vérification et de la validation des utilisateurs et qui enregistre dans une session PHP les données utilisateurs. La propriété *UserClass* est initialisée comme étant de type *Application.BlogUser*, ceci indique au module *user* de chercher une classe *BlogUser* dans le dossier *protected* (rappellez-vous que l'alias *Application* fait référence au dossier *protected*) et d'utiliser cette classe pour conserver les données utilisateurs dans une session.

Comme vous pourrez le constater dans les sections suivantes, dans les différents contrôles et pages, nous pourrons utiliser *$this->User* pour accéder à l'objet qui contient les informations de l'utilisateur actuellement connecté.

Ci-dessous les détails de l'implémentation de la classe *BlogUser*. Remarquez que les *Active Record* sont utilisés pour exécuter une requête. Par exemple, nous utilisons *UserRecord::finder()->findByPk($username)* pour chercher la valeur de *$username* dans la table *users* et ceci par la clé primaire.

```
// Include TDbUserManager.php file which defines TDbUser
Prado::using('System.Security.TDbUserManager');

/**
 * La classe BlogUser.
 * BlogUser représente les données utilisateurs à conserver en session.
 * L'implémentation par défaut conserve le nom et le rôle de l'utilisateur.
 */
class BlogUser extends TDbUser
{
    /**
     * Créer un objet de type BlogUser basé sur le nom de l'utilisateur.
     * Cette méthode est requise par TDbUser. Cet objet vérifie si l'utilisateur
     * est bien présent en base de données. Si oui, un objet BlogUser
     * est créé et initialisé.
     * @param string le nom de l'utilisateur
     * @return l'objet BlogUser, null si le nom de l'utilisateur est invalide.
     */
    public function createUser($username)
    {
        // utilise l'Active Record  UserRecord pour chercher l'utilisateur username
        $userRecord=UserRecord::finder()->findByPk($username);
        if($userRecord instanceof UserRecord) // si trouvé
        {
            $user=new BlogUser($this->Manager);
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
            $user->Name=$username;  // enregistre le nom de l'utilisateur
            $user->Roles=($userRecord->role==1?'admin':'user'); // et son rôle
            $user->IsGuest=false;   // l'utilisateur n'est pas un invité
            return $user;
        }
        else
            return null;
    }

    /**
     * Vérifie que le nom d'utilisateur et son mot de passe sont correct.
     * Cette méthode est requise par TDbUser.
     * @param string le nom de l'utilisateur
     * @param string le mot de passe
     * @return boolean en fonction de la validité de la vérification.
     */
    public function validateUser($username,$password)
    {
  // utilise l'Active Record UserRecord pour vérifier le nom d'utilisateur couplé au mot de passe.
        return UserRecord::finder()->findBy_username_AND_password($username,$password)!==null;
    }

    /**
     * @return boolean indiquant si l'utilisateur est un administrateur.
     */
    public function getIsAdmin()
    {
        return $this->isInRole('admin');
    }
}
```

IV-C - Création de la page 'LoginUser'

La page *LoginUser* affiche un formulaire et gère l'authentification de l'utilisateur. Comme décrit dans **authentification et autorisation**, le navigateur est automatiquement redirigé vers la page *LoginUser* quand un utilisateur essaye d'accéder à une page protégée, telle que la page d'administration des utilisateurs.

Le processus de la page *LoginUser* est similaire à celui de la page *Contact*:

1. Quand un utilisateur accède à la page *LoginUser*, un formulaire est affiché;
2. L'utilisateur remplit les champs, nom de l'utilisateur et mot de passe et clique sur le bouton "envoyer";
3. La classe *LoginUser* reçoit l'évènement "login" et lance la séquence d'authentification;
4. Si le nom d'utilisateur et le mot de passe sont corrects, le système l'enregistre en session et le redirige vers la page protégée demandée. Dans le cas contraire, un message "mot de passe invalide" est affiché.

Nous créons les deux fichiers *protected/pages/users/LoginUser.page* et *protected/pages/users/LoginUser.php* qui enregistre le gabarit et la classe respectivement.

Création du gabarit

Ci-après est affiché le gabarit pour *LoginUser*. Comme vous pouvez le constater, la page contient un champ de saisie pour le nom de l'utilisateur et un autre pour le mot de passe. Le nom de l'utilisateur est requis, ce que le validateur *TRequiredFieldValidator* contrôle. La validité du mot de passe est assurée par le validateur **TCustomValidator** qui fait un appel à la méthode *validateUser()* de la classe. La page contient aussi un bouton "envoyer" qui fait un appel à *loginButtonClicked()* quand il est activé.

```
<%@ Title="Mon Blog - Login" %>

<com:TContent ID="Main">

<h1>Connexion</h1>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
<span>Votre nom:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Username"
    ErrorMessage="Veuillez indiquer votre nom."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Username" />

<br/>
<span>Mot de passe:</span>
<com:TCustomValidator
    ControlToValidate="Password"
    ErrorMessage="vous avez saisi un mot de passe invalide."
    Display="Dynamic"
    OnServerValidate="validateUser" />
<br/>
<com:TTextBox ID="Password" TextMode="Password" />

<br/>
<com:TButton Text="Envoyer" OnClick="loginButtonClicked" />

</com:TContent>
```

Création de la classe

Tout comme la page *Contact*, la page *LoginUser* a aussi besoin d'un fichier de classe qui implémente les évènements générés dans le fichier gabarit. Ici, nous avons besoin de deux méthodes : *validateUser()* et *loginButtonClicked()*. Dans *validateUser()*, nous utilisons le gestionnaire d'authentification pour vérifier si le nom d'utilisateur et le mot de passe sont valides. Si c'est le cas, le gestionnaire d'authentification créé automatiquement une session utilisateur avec les données correspondantes.

```
class LoginUser extends TPage
{
    /**
     * Vérifie la validité du nom d'utilisateur et du mot de passe.
     * Cette méthode implémente l'évènement <tt>OnServerValidate</tt> du validateur <tt>TCustomValidator</tt>.
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function validateUser($sender,$param)
    {
        $authManager=$this->Application->getModule('auth');
        if(!$authManager->login($this->Username->Text,$this->Password->Text))
            $param->IsValid=false;  // indique au validateur que la validation à échoué
    }

    /**
     * Rédirige le navigateur vers l'URL originellement demandée si la validation est Ok.
     * Cette méthode implémente l'évènement <tt>OnClick</tt> du bouton "envoyer".
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function loginButtonClicked($sender,$param)
    {
        if($this->Page->IsValid)  // toutes les validations sont ok ?
        {
            // récupère l'URL de la page protégée qui avait été demandée par l'utilisateur
            $url=$this->Application->getModule('auth')->ReturnUrl;
            if(empty($url))  // l'utilisateur à accéder à la page de connexion directement
                $url=$this->Service->DefaultPageUrl;
            $this->Response->redirect($url);
        }
    }
}
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Test

Nous avons donc créé la page *LoginUser*. Nous pouvons la tester en naviguant à l'URL *http://hostname/blog/index.php?page=users.LoginUser*. Rappellez-vous que la dans la section *Création de la base*, nous avons déjà créé deux comptes utilisateurs (nom d'utilisateur/mot de passe) *admin/demo* et *demo/demo*. Nous pouvons donc les utiliser pour tester notre page de connexion.

My PRADO Blog

Login

Username:

Password:

Login

My PRADO Blog

Login

Username:

admin

Password: Your entered an invalid pas

Login

Ajout des liens de connexion/déconnexion à notre gabarit principal

Pour permettre à l'utilisateur d'accéder directement aux pages de connexion/déconnexion, nous modifions le gabarit principal *MainLayout*. En particulier, nous ajoutons un lien vers la page *LoginUser*. Nous ajoutons aussi un lien "se déconnecter" qui permet à l'utilisateur de se déconnecter.

Nous modifions le pied de page de notre gabarit principal *MainLayout*. La visibilité des liens vers "se connecter" et "se déconnecter" dépend du statut de l'utilisateur. Si l'utilisateur n'est pas encore connecté, ie: *$this->User->IsGuest* est vrai, alors le lien "se connecter" est visible tandis que le lien "se déconnecter" ne l'est pas et inversement s'il est connecté.

```
<div id="footer">
<com:THyperLink Text="Se connecter"
    NavigateUrl="<%= $this->Service->constructUrl('users.LoginUser') %>"
    Visible="<%= $this->User->IsGuest %>" />

<com:TLinkButton Text="Se déconnecter"
    OnClick="logoutButtonClicked"
    Visible="<%= !$this->User->IsGuest %>" />

<br/>
<%= PRADO::poweredByPrado() %>
</div>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Vu que le lien "se déconnecter" génère l'évènement *OnClick* avec comme nom d'évènement *logoutButtonClicked()*, nous devons modifier le fichier de classe de *MainLayout* comme ci-dessous :

```
class MainLayout extends TTemplateControl
{
    /**
     * Déconnecte un utilisateur.
     * Cette méthode répond à l'évènement OnClick du lien "se déconnecter".
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function logoutButtonClicked($sender,$param)
    {
        $this->Application->getModule('auth')->logout();
        $url=$this->Service->constructUrl($this->Service->DefaultPage);
        $this->Response->redirect($url);
    }
}
```

Maintenant si nous visitons n'importe quelle page de notre blog, nous verrons apparaitre un lien en pied de page. Le lien affiche "se connecter" si nous ne sommes pas connectés et "se déconnecter" dans le cas contraire. Si nous cliquons sur le lien "se déconnecter", nous sommes redirigés vers la page d'accueil et le lien "se connecter" apparait indiquant que nous ne sommes plus connectés.

IV-D - Création de la page nouvel utilisateur 'NewUser'

La page *NewUser* est fournie à l'administrateur pour créer des nouveaux comptes utilisateurs. Elle doit afficher un formulaire qui permet la saisie des informations d'un nouveau compte. Tel que défini dans la base de données, nous devons prévoir la saisie des informations suivantes :

- *username* - string, pseudo de l'utilisateur, obligatoire et unique
- *email* - string, email, obligatoire et unique
- *password* - string, mot de passe, obligatoire
- *role* - integer, rôle, obligatoire (0 ou 1)
- *first_name* - string, prénom, optionnel
- *last_name* - string, nom, optionnel

Nous créons deux fichiers, *protected/pages/users/NewUser.page* et *protected/pages/users/NewUser.php* qui contiendront respectivement le gabarit et la classe.

Création du gabarit

En fonction de l'analyse précédente, nous créons le gabarit comme ci-dessous :

```
<%@ Title="Mon Blog - Nouvel utilisateur" %>

<com:TContent ID="Main">

<h1>Création nouvel utilisateur</h1>

<span>Pseudo:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Username"
    ErrorMessage="Veuillez indiquer un pseudo."
    Display="Dynamic" />
<com:TCustomValidator
    ControlToValidate="Username"
    ErrorMessage="Désolé, le pseudo choisi est déjà utilisé. Veuillez en saisir un autre."
    OnServerValidate="checkUsername"
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Username" />

<br/>
<span>Mot de passe:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Password"
    ErrorMessage="Veuillez indiquer un mot de passe."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Password" TextMode="Password" />

<br/>
<span>Confirmation mot de passe:</span>
<com:TCompareValidator
    ControlToValidate="Password"
    ControlToCompare="Password2"
    ErrorMessage="Différence entre le mot de passe et la confirmation."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Password2" TextMode="Password" />

<br/>
<span>Email:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Email"
    ErrorMessage="Veuillez indiquer votre email."
    Display="Dynamic" />
<com:TEmailAddressValidator
    ControlToValidate="Email"
    ErrorMessage="Vous avez indiqué un mot de passe invalide."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Email" />

<br/>
<span>Rôle:</span>
<br/>
<com:TDropDownList ID="Role">
    <com:TListItem Text="Utilisateur standard" Value="0" />
    <com:TListItem Text="Administrateur" Value="1" />
</com:TDropDownList>

<br/>
<span>Prénom:</span>
<br/>
<com:TTextBox ID="FirstName" />

<br/>
<span>Nom:</span>
<br/>
<com:TTextBox ID="LastName" />

<br/>
<com:TButton Text="Ajouter" OnClick="createButtonClicked" />

</com:TContent>
```

Le gabarit est très proche du gabarit de la page *Contact* et de la page *LoginUser*. Il consiste principalement en deux champs de saisie et de plusieurs validateurs. Certains champs de saisie sont associés à plusieurs validateurs vu qu'il est nécessaire de vérifier plusieurs règles.

Création du fichier de classe

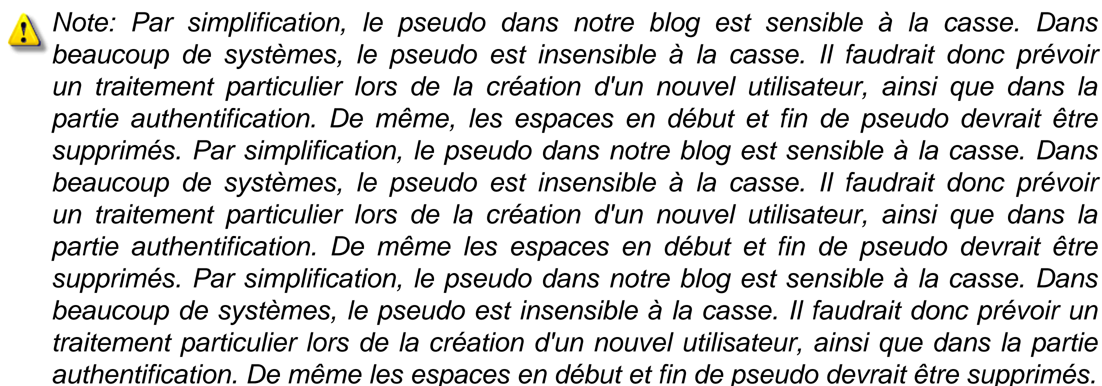
En fonction du gabarit précédent, nous constatons que nous avons besoin d'une classe qui implémente deux gestionnaires d'évènements : *checkUsername()* (appellé par le premier validateur dans l'évènement

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

OnServerValidate) et *createButtonClicked()* (appellé par l'évènement *OnClick* du bouton "create"). ainsi, nous écrirons la classe comme ci-dessous :

```
class NewUser extends TPage
{
    /**
     * Vérifie si le nom d'utilisateur existe dans la base de données.
     * Cette méthode répond à l'évènement OnServerValidate du validateur username.
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function checkUsername($sender,$param)
    {
        // valide si l'utilisateur existe
        $param->IsValid=UserRecord::finder()->findByPk($this->Username->Text)===null;
    }

    /**
     * Créer un nouveau compte utilisateur si tous les champs sont valides.
     * Cette méthode répond à l'évènement OnClick du bouton "create".
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function createButtonClicked($sender,$param)
    {
        if($this->IsValid)  // si toutes les validations sont ok
        {
            // rempli l'objet UserRecord avec les données saisies
            $userRecord=new UserRecord;
            $userRecord->username=$this->Username->Text;
            $userRecord->password=$this->Password->Text;
            $userRecord->email=$this->Email->Text;
            $userRecord->role=(int)$this->Role->SelectedValue;
            $userRecord->first_name=$this->FirstName->Text;
            $userRecord->last_name=$this->LastName->Text;

            // l'enregistre dans la base de données par la méthode save de l'Active Record
            $userRecord->save();

            // redirige l'utilisateur vers la page d'accueil
   $this->Response->redirect($this->Service->DefaultPageUrl);
        }
    }
}
```

Dans le code précédent, l'appel à la méthode *save()* insère un enregistrement dans la table *users*. Cette fonctionnalité est fournie par l'objet **Active Record**.



Note: Par simplification, le pseudo dans notre blog est sensible à la casse. Dans beaucoup de systèmes, le pseudo est insensible à la casse. Il faudrait donc prévoir un traitement particulier lors de la création d'un nouvel utilisateur, ainsi que dans la partie authentification. De même, les espaces en début et fin de pseudo devrait être supprimés. Par simplification, le pseudo dans notre blog est sensible à la casse. Dans beaucoup de systèmes, le pseudo est insensible à la casse. Il faudrait donc prévoir un traitement particulier lors de la création d'un nouvel utilisateur, ainsi que dans la partie authentification. De même les espaces en début et fin de pseudo devrait être supprimés. Par simplification, le pseudo dans notre blog est sensible à la casse. Dans beaucoup de systèmes, le pseudo est insensible à la casse. Il faudrait donc prévoir un traitement particulier lors de la création d'un nouvel utilisateur, ainsi que dans la partie authentification. De même les espaces en début et fin de pseudo devrait être supprimés.

Test

Pour tester la page *NewUser*, il suffit de naviguer à l'URL *http://hostname/blog/index.php?page=users.NewUser*. Vous devriez voir apparaitre la page suivante. Essayez de saisir différentes informations et remarquez comment

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

les données sont validées. Si toutes les règles sont valides, nous devrions avoir inséré un nouvel utilisateur et être redirigés vers la page d'accueil.

My PRADO Blog

Create New User

Username:

Password:

Re-type Password:

Email Address:

Role:

Normal User

First Name:

Last Name:

Create

Login

Ajout de la vérification des droits d'accès

Durant le test, vous vous êtes peut-être demandé : Est-ce que la page *NewUser* ne devrait être accessible qu'aux administrateurs ? Oui, ceci est dénommé **autorisation**. Nous allons maintenant décrire comment ajouter cette vérification d'accès à la page *NewUser*.

Une façon simple serait de vérifier dans le code de la classe si *$this->User->IsAdmin* est vrai, dans le cas contraire, une redirection vers la page de connexion *LoginUser* serait faite.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

PRADO propose une approche complémentaire de vérification des droits. Pour ce faire, nous devons utiliser un fichier de **configuration de page**. Créer un fichier *protected/pages/users/config.xml* avec le contenu suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <authorization>
    <allow pages="NewUser" roles="admin" />
    <deny users="?" />
  </authorization>
</configuration>
```

Le fichier de configuration de page contient les règles d'accès aux pages contenues dans le dossier *protected/pages/users*. Ce fichier indique que la page *NewUser* peut être vue par les utilisateurs dont le rôle est "admin" (concernant le mot "admin" voir BlogUser.createUser()), et toutes les autres pages de ce dossier interdites aux utilisateurs anonymes (*users="?"* signifie utilisateur anonyme).

Dorénavant, si nous naviguons à la page *NewUser* en tant qu'anonyme, nous serons redirigés vers la page *LoginUser*. Si notre connexion est acceptée, nous serons redirigés en retour vers la page *NewUser*.

Astuce: Le fichier de configuration de pages peut contenir d'autres éléments que les règles d'autorisations. Par exemple, il pourrait inclure un ***module*** *tout comme nous l'avons fait pour le fichier de* ***configuration de l'application****. Dans une application PRADO, chaque dossier de pages peut contenir un fichier de configuration de pages qui s'applique à tous les fichiers du dossier ainsi qu'aux sous dossiers.*

IV-E - Création de la page de mise à jour des utilisateurs 'EditUser'

La page *EditUser* ressemble beaucoup à la page *NewUser*. La différence principale est le fait que lorsque la page *EditUser* est requise, les champs sont renseignés avec les données de l'utilisateur en cours. Une autre différence est que la page *EditUser* est accessible à un utilisateur standard.

Pour définir quels sont les comptes qui peuvent être modifiés, nous allons utiliser les règles suivantes :

- Si l'utilisateur actuel est un administrateur, il peut modifier n'importe quel compte utilisateur en spécifiant le pseudo de l'utilisateur dans l'URL sous la forme ?username='le nom'. Par exemple : *http://hostname/blog/index.php?page=users.EditUser&username=demo.*
- Si l'utilisateur actuel est un administrateur et qu'il n'a pas précisé d'username, ce sont les informations de l'administrateur qui seront mise à jour.
- Si l'utilisateur actuel est un utilisateur standard, seules les données de son compte seront accessibles et il ne pourra pas modifier son rôle.

Nous créons deux fichiers *protected/pages/users/EditUser.page* et *protected/pages/users/EditUser.php* qui contiendront respectivement le gabarit et la classe.

Création du gabarit

Comme vous avez pu le deviner, la page *EditUser* est fortement ressemblante à la page *NewUser*. En dehors du titre de la page et du bouton "envoyer", il y a trois différences principales.

- Le champ de saisie "username" est remplacé par un contrôle **TLabel** vu qu'il n'est pas autorisé de modifier son pseudo;
- Le validateur pour le champ "password" est supprimé. Si l'utilisateur ne fournit pas de mot de passe durant la modification, cela indique que l'utilisateur ne veut pas en changer.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- Le champ "role" est entouré d'un *TControl*, ce qui nous permet de la rendre visible ou invisible en fonction de rôle de l'utilisateur connecté. Si l'utilisateur n'est pas un administrateur, le champ "role" ne sera pas affiché. Les utilisateurs standard n'ont pas le droit de modifier leur rôle.

```
<%@ Title="Mon blog - Mise à jour utilisateur" %>

<com:TContent ID="Main">

<h1>Mise à jour utilisateur</h1>

<span>Pseudo:</span>
<com:TLabel ID="Username" />

<br/>
<span>Mot de passe:</span>
<br/>
<com:TTextBox ID="Password" TextMode="Password" />

<br/>
<span>Confirmation mot de passe:</span>
<com:TCompareValidator
    ControlToValidate="Password"
    ControlToCompare="Password2"
    ErrorMessage="Différence entre le mot de passe et la confirmation."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Password2" TextMode="Password" />

<br/>
<span>Email:</span>
<com:TRequiredFieldValidator
    ControlToValidate="Email"
    ErrorMessage="Veuillez indiquer votre email."
    Display="Dynamic" />
<com:TEmailAddressValidator
    ControlToValidate="Email"
    ErrorMessage="Vous avez indiqué un mot de passe invalide."
    Display="Dynamic" />
<br/>
<com:TTextBox ID="Email" />

<com:TControl Visible="<%= $this->User->IsAdmin %>">
<br/>
<span>Role:</span>
<br/>
<com:TDropDownList ID="Role">
    <com:TListItem Text="Utilisateur standard" Value="0" />
    <com:TListItem Text="Administrateur" Value="1" />
</com:TDropDownList>
</com:TControl>

<br/>
<span>Prénom:</span>
<br/>
<com:TTextBox ID="FirstName" />

<br/>
<span>Nom:</span>
<br/>
<com:TTextBox ID="LastName" />

<br/>
<com:TButton Text="Enregistrer" OnClick="saveButtonClicked" />

</com:TContent>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création du fichier de classe

En suivant les indications du gabarit, nous devons écrire une page de classe qui initialise les champs avec les données de l'utilisateur. De plus, la classe doit implémenter la méthode *saveButtonClicked()* appelée par l'évènement *OnClick* du bouton "Enregistrer".

```
class EditUser extends TPage
{
    /**
     * Initialise les champs avec les données de l'utilisateur.
     * Cette méthode est appelée par le framework lorsque la page est initialisée.
     * @param mixed param : paramètres de l'évènement
     */
    public function onInit($param)
    {
        parent::onInit($param);
        if(!$this->IsPostBack)  // est-ce que c'est le premier appel à la page
        {
            // Lit les informations de l'utilisateur. C'est équivalent à :
            // $userRecord=$this->getUserRecord();
            $userRecord=$this->UserRecord;

            // Rempli les contrôles avec les données de l'utilisateur
            $this->Username->Text=$userRecord->username;
            $this->Email->Text=$userRecord->email;
            $this->Role->SelectedValue=$userRecord->role;
            $this->FirstName->Text=$userRecord->first_name;
            $this->LastName->Text=$userRecord->last_name;
        }
    }

    /**
     * Enregistre les modifications si tous les validateurs sont Ok.
     * Cette méthode répond à l'évènement OnClick du bouton "Enregistrer".
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function saveButtonClicked($sender,$param)
    {
        if($this->IsValid)  // toutes les validations Ok ?
        {
            // Lit les informations de l'utilisateur.
            $userRecord=$this->UserRecord;

            // Enresgistre les valeurs dans les champs de la BDD
            $userRecord->username=$this->Username->Text;
            // mets à jour le mot de passe s'il n'est pas vide
            if(!empty($this->Password->Text))
                $userRecord->password=$this->Password->Text;
            $userRecord->email=$this->Email->Text;
            // mets à jour le rôle si l'utilisateur actuel est un administrateur
            if($this->User->IsAdmin)
                $userRecord->role=(int)$this->Role->SelectedValue;
            $userRecord->first_name=$this->FirstName->Text;
            $userRecord->last_name=$this->LastName->Text;

            // enregistre les modifications dans la BDD
            $userRecord->save();

            // redirige vers la page d'accueil
   $this->Response->redirect($this->Service->DefaultPageUrl);
        }
    }

    /**
     * Retourne l'utilisateur qui doit être mis à jour.
     * @return UserRecord l'utilisateur qui doit être modifié.
     * @throws THttpException si l'utilisateur n'existe pas.
     */
    protected function getUserRecord()
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
    {
        // l'utilisateur à modifié est l'utilisateur actuellement connecté
        $username=$this->User->Name;
        // si la variable GET 'username' n'est pas vide et que l'utilisateur actuel
        // est un administrateur, nous utilisons la variable GET à la place
        if($this->User->IsAdmin && $this->Request['username']!==null)
            $username=$this->Request['username'];

        // lit les données de l'utilisateur par Active Record
        $userRecord=UserRecord::finder()->findByPk($username);
        if(!($userRecord instanceof UserRecord))
            throw new THttpException(500,'Username is invalid.');
        return $userRecord;
    }
}
```

Astuce: La méthode onInit() est appelée par PRADO lors du ***cycle de vie de la page****. Les autres méthodes couramment surchargées sont onPreInit(), onLoad() et onPreRender().*

Test

Pour tester la page *EditUser*, rendons-nous à l'URL *http://hostname/blog/index.php?page=users.EditUser&username=demo*. Il vous sera peut-être demandé de vous authentifier auparavant si vous n'êtes pas déjà connecté. Essayez de vous connecter avec différents comptes (ie: *admin/demo*, *demo/demo*) et remarquez comment la page évolue différemment.

IV-F - Création de la page d'administration des utilisateurs 'AdminUser'

La page *AdminUser* affiche la liste de tous les comptes utilisateurs, ainsi l'administrateur peut effectuer les tâches de maintenance. Par simplification, les tâches administratives pour notre blog seront la mise à jour des utilisateurs et la suppression.

Nous allons lister les utilisateurs dans une table HTML. Chaque ligne correspondra à un compte utilisateur, les colonnes suivantes seront affichées :

- Pseudo - affiche le pseudo de l'utilisateur. Dans chaque cellule un lien sera affiché qui nous dirigera vers la page EditUser.
- Email - affiche l'email.
- Administrateur - indique si le compte est celui d'un administrateur.
- Commande - affiche une colonne de bouton "supprimer". En cliquant sur un de ces boutons, la suppression du compte sera effectuée.

Nous créons deux fichiers *protected/pages/users/AdminUser.page* et *protected/pages/users/AdminUser.php* qui contiendront respectivement le gabarit et la classe.

Création du gabarit

Nous allons utiliser un contrôle **TDataGrid** pour afficher les données. Suivant l'analyse précédente, nous allons configurer quatre colonnes :

- **THyperLinkColumn** affiche le pseudo. L'URL sera construite suivant les instructions de la propriété DataNavigateUrlFormatString.
- **TBoundColumn** affiche l'email.
- **TCheckBoxColumn** utilise des cases à cocher pour indiquer si le compte est un compte administrateur.
- **TButtonColumn** affiche un bouton "Supprimer".

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Le gabarit complet est affiché ci-après :

```
<%@ Title="Mon Blog - Administration des comptes utilisateurs" %>

<com:TContent ID="Main">

<h1>Administration des comptes utilisateurs</h1>

<a href="<%= $this->Service->constructUrl('users.NewUser')%>">Créer un nouvel utilisateur</a>
<br/>

<com:TDataGrid ID="UserGrid"
    DataKeyField="username"
    AutoGenerateColumns="false"
    OnDeleteCommand="deleteButtonClicked">

    <com:THyperLinkColumn
        HeaderText="Pseudo"
        DataTextField="username"
        DataNavigateUrlField="username">
        <prop:DataNavigateUrlFormatString>#
          $this->Service->constructUrl('users.EditUser',array('username'=>{0}))
        </prop:DataNavigateUrlFormatString>
    </com:THyperLinkColumn>

    <com:TBoundColumn
        HeaderText="Email"
        DataField="email" />

    <com:TCheckBoxColumn
        HeaderText="Administrateur"
        DataField="role" />

    <com:TButtonColumn
        HeaderText="Commande"
        Text="Supprimer"
        ButtonType="PushButton"
        CommandName="delete" />

</com:TDataGrid>

</com:TContent>
```

Création du fichier de classe

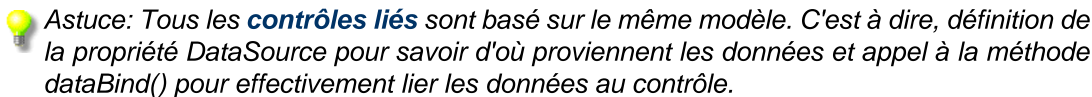
Dans le gabarit précédent, le bouton *OnDeleteCommand* déclenche l'évènement *deleteButtonClicked()* que nous devons implémenter dans le fichier de classe. De plus, la grille de données doit être renseignée avec les informations utilisateurs lorsque la page est initialisée. Nous écrivons donc notre fichier de classe comme ci-dessous :

```
class AdminUser extends TPage
{
    /**
     * Remplis la grille avec la liste des utilisateurs.
     * Cette méthode est appelée lors de l'initialisation de la page.
     * @param mixed param : paramètres de l'évènement
     */
    public function onInit($param)
    {
        parent::onInit($param);
        // lit tout les comptes utilisateurs
        $this->UserGrid->DataSource=UserRecord::finder()->findAll();
        // et les associes à la grille
        $this->UserGrid->dataBind();
    }

    /**
     * Supprime un compte utilisateur.
     * Cette méthode répond à l'évènement OnDeleteCommand.
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
     * @param mixed sender : celui qui a généré l'évènement
     * @param mixed param : paramètres de l'évènement
     */
    public function deleteButtonClicked($sender,$param)
    {
        // récupère l'identifiant du bouton sur lequel on a cliqué
        $item=$param->Item;
        // récupère auprès de la grille la clé primaire correspondante à l'identifiant
        $username=$this->UserGrid->DataKeys[$item->ItemIndex];
        // supprime le compte utilisateur en utilisant la clé primaire
        UserRecord::finder()->deleteByPk($username);
    }
}
```

Dans le code précédent, la méthode *deleteButtonClicked()* est appelée quand on clique sur le bouton "Supprimer". Pour savoir à quelle ligne appartenait le bouton, nous utilisons la propriété *Item.ItemIndex* du paramètre de l'évènement. Avec cet index, nous recherchons quelle est la clé primaire de la ligne grâce à la propriété *DataKeys*.



Astuce: Tous les ***contrôles liés*** *sont basé sur le même modèle. C'est à dire, définition de la propriété DataSource pour savoir d'où proviennent les données et appel à la méthode dataBind() pour effectivement lier les données au contrôle.*

Ajout de la vérification des droits d'accès

Vu que seuls les administrateurs doivent pouvoir accéder à la page AdminUser, nous devons modifier notre fichier de configuration de page *protected/pages/users/config.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <authorization>
    <allow pages="NewUser,AdminUser" roles="admin" />
    <deny users="?" />
  </authorization>
</configuration>
```

Test

Pour tester notre page *AdminUser*, nous naviguons à l'adresse *http://hostname/blog/index.php?page=users.AdminUser*. Il peut vous être demandé de vous connecter en tant qu'administrateur auparavant si ce n'est déjà fait. Le résultat suivant apparaitra :

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

Manage User Accounts

Username	Email	Administrator	Command
admin	admin@example.com	☑	Delete
demo	demo@example.com	☐	Delete

Logout

V - 4ième jour: Gestion des Messages

V-A - Vue d'ensemble de la gestion des messages

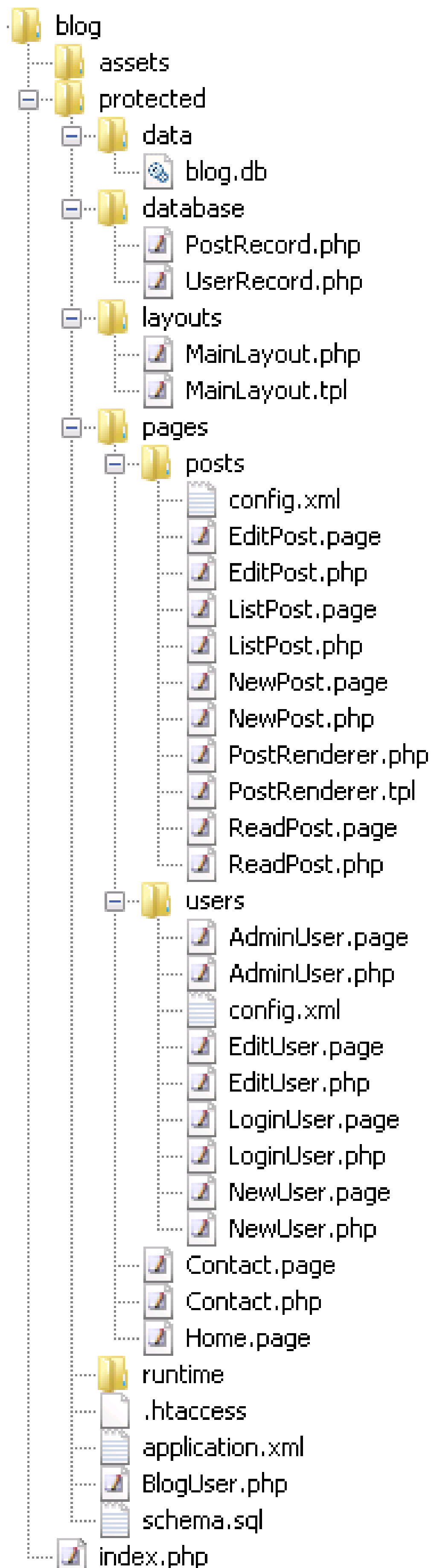
Dans cette section, nous allons créer les pages correspondantes à la gestion des messages. En particulier, nous mettrons en place les quatre opérations de base (Création-Lecture-Modification-Suppression) (CRUD:Create-Retrieve-Update-Delete).

Nous allons créer les nouvelles pages dans le dossier *protected/pages/posts* créé à cet effet.

- *ListPost* affiche la liste des messages triés par ordre de date décroissante.
- *ReadPost* affiche le détail d'un message.
- *NewPost* permet aux utilisateurs enregistrés de créer un nouveau message.
- *EditPost* permet à l'auteur et aux administrateurs de modifier un message.

Après avoir fini cette section, nous devrions obtenir l'arborescence suivante :

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

V-B - Création de la page d'affichage des messages 'ListPost'

La page *ListPost* affiche les derniers messages sous forme de liste. S'il y a trop de messages, ils seront affichés dans différentes pages.

Avant que nous ne passions à l'implémentation, nous voudrions que notre page d'accueil pointe vers la page à venir *ListPost*, ceci dans le but d'afficher la liste des derniers messages dès qu'un utilisateur se connecte au site. Pour cela, nous allons modifier le fichier de configuration de l'application *protected/application.xml* de cette manière :

```
......
<services>
  <service id="page" class="TPageService" DefaultPage="posts.ListPost">
    <pages MasterClass="Application.layouts.MainLayout" />
  </service>
</services>
```

Nous alons maintenant créer le gabarit et le fichier de classe pour notre page *ListPost* : *protected/pages/posts/ListPost.page* et *protected/pages/posts/ListPost.php*.

Création du gabarit

Pour satisfaire les fonctionnalités de notre page *ListPost*, nous allons utiliser deux contrôles dans notre gabarit.

- **TRepeater**: ce contrôle permet d'afficher principalement une liste d'éléments. La présentation de chacun ce ces éléments peut être défini soit par un gabarit interne, soit par un gabarit externe (choix que nous avons fait).
- **TPager**: ce contrôle permet de faire la pagination d'une liste d'éléments. Il interagit avec l'utilisateur pour définir quelle page doit être affiché dans un **contrôle de liste** (ie: *TListBox*) ou dans un **contrôle de données** (ie: *TRepeater*).

Ci-dessous le contenu du gabarit :

```
<%@ Title="Mon Blog" %>

<com:TContent ID="Main">

<com:TRepeater ID="Repeater"
 ItemRenderer="Application.pages.posts.PostRenderer"
 AllowPaging="true"
 AllowCustomPaging="true"
 PageSize="5"
 />

<com:TPager ControlToPaginate="Repeater" OnPageIndexChanged="pageChanged" />

</com:TContent>
```

Dans la partie répétée *TRepeater*, nous indiquons que l'affichage du contenu est délégué à l'élément *PostRenderer* que nous allons créer après. Pour permettre à PRADO de trouver cette classe, nous fournissons l'espace de noms complet *Application.pages.posts.PostRenderer*, qui correspond au fichier *protected/pages/posts/PostRenderer.php*.

Nous définissons aussi quelques propriétés complémentaires du *TRepeater* pour activer la pagination. Et nous définissons la propriété *ControlToPaginate* du *TPager* afin qu'il sache quelle est la zone répetée à paginer.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création du fichier de classe

En fonction du gabarit précédent, nous pouvons voir que notre fichier de classe doit implémenter un gestionnaire d'évènement pour *pageChanged()* (déclenché par *OnPageIndexChanged* du *TPager*). Nous devons aussi remplir les données qui apparaitront dans le *TRepeater*. Ci-dessous le source complet du fichier de classe :

```
class ListPost extends TPage
{
 /**
  * Initialise le TRepeater.
  * Cette méthode est appelé par le framework lors de l'initialisation de la page
  * @param mixed param : paramètres de l'évènement
  */
 public function onInit($param)
 {
  parent::onInit($param);
  if(!$this->IsPostBack)  // la page est chargée pour la première fois ?
  {
   // récupère le nombre total de messages
   $this->Repeater->VirtualItemCount=PostRecord::finder()->count();
   // rempli le TRepeater avec les données
   $this->populateData();
  }
 }

 /**
  * Gestionnaire d'évènement pour OnPageIndexChanged du TPager.
  * Cette méthode est appelée lors du changement de page
  */
 public function pageChanged($sender,$param)
 {
  // change l'index de la page courante par le nouvel index
  $this->Repeater->CurrentPageIndex=$param->NewPageIndex;
  // rempli de nouveau le TRepeater
  $this->populateData();
 }

 /**
  * détermine quelle page doit être affichée et remplie
  * TRepeater avec les données lues
  */
 protected function populateData()
 {
  $offset=$this->Repeater->CurrentPageIndex*$this->Repeater->PageSize;
  $limit=$this->Repeater->PageSize;
  if($offset+$limit>$this->Repeater->VirtualItemCount)
   $limit=$this->Repeater->VirtualItemCount-$offset;
  $this->Repeater->DataSource=$this->getPosts($offset,$limit);
  $this->Repeater->dataBind();
 }

 /**
  * lis les données à partir de la base de données en utilisant les fonctionnalités offset et limit.
  */
 protected function getPosts($offset, $limit)
 {
  // construit les critères de la requête
  $criteria=new TActiveRecordCriteria;
  $criteria->OrdersBy['create_time']='desc';
  $criteria->Limit=$limit;
  $criteria->Offset=$offset;
  // lit les messages en fonction des critères précédents
  return PostRecord::finder()->withAuthor()->findAll($criteria);
 }
}
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création du PostRenderer

Nous devons toujours créer la classe *PostRenderer*. Elle définit la manière dont sera affichée chaque ligne de notre *TRepeater*. Nous la créons en tant que gabarit de contrôle, ce qui nous permet d'utiliser notre système de gabarit. Le fichier de gabarit ainsi que notre fichier de classe seront sauvegardés respectivement sous *PostRenderer.tpl* et *PostRenderer.php* dans le dossier *protected/pages/posts*.

Création du gabarit pour *PostRenderer*

Le gabarit définit la présentation des différentes informations d'un message : titre, nom, heure, contenu. Nous lions le titre à la page *ReadPost* qui affiche le détail du message.

L'expression *$this->Data* fait référence aux données provenant du *TRepeater*. Dans notre cas, c'est un objet de type *PostRecord*. Remarquez comment nous accédons au nom de l'auteur du message par *$this->Data->author->username*.

```
<div class="post-box">
<h3>
<com:THyperLink Text="<%# $this->Data->title %>"
 NavigateUrl="<%# $this->Service->constructUrl('posts.ReadPost',array('id'=>$this->Data->post_id)) %>" />
</h3>

<p>
Auteur:
<com:TLiteral Text="<%# $this->Data->author->username %>" /><br/>
Heure:
<com:TLiteral Text="<%# date('m/d/Y h:m:sa', $this->Data->create_time) %>" />
</p>

<p>
<com:TLiteral Text="<%# $this->Data->content %>" />
</p>
</div>
```

Création du fichier de classe pour *PostRenderer*

Notre classe est très simple, elle hérite de *TRepeaterItemRenderer* et ne contient aucun autre code.

```
class PostRenderer extends TRepeaterItemRenderer
{
}
```

Test

Pour tester la page *ListPost*, naviguons à l'URL *http://hostname/blog/index.php* (rappellez-vous, nous avons défini *ListPost* comme étant notre page d'accueil). Nous devrions obtenir le résultat suivant. vu que nous n'avons qu'un seul message pour le moment, le contrôle de pagination n'apparait pas. Plus tard, quand nous aurons fini la page *NewPost*, nous pourrons ajouter des messages et revenir ici pour tester notre contrôle de pagination.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

first post

Author: admin
Time: 04/04/2007 01:04:22pm

this is my first post

Login

V-C - Création de la page détail d'un message 'ReadPost'

La page *ReadPost* affiche le détail d'un message. Pour les utilisateurs autorisés, un lien sera disponible permettant de modifier ou de supprimer le message.

Nous créons deux fichiers *protected/pages/posts/ReadPost.page* et *protected/pages/posts/ReadPost.php* qui contiendront respectivement notre gabarit et notre classe.

V-C-1 - Création du gabarit

Le gabarit de *ReadPost* est très proche du gabarit de *PostRenderer*, chacun d'eux affiche le détail d'un message. La différence est que la page *ReadPost* doit afficher deux boutons, permettant aux utilisateurs autorisés de modifier ou supprimer le message.

```
<com:TContent ID="Main">

<h2>
<com:TLiteral Text="<%= $this->Post->title %>" />
</h2>

<com:TControl Visible="<%= $this->canEdit() %>">
 <a href="<%= $this->Service->constructUrl('posts.EditPost',array('id'=>$this->Post-
>post_id))%>">Modifier</a> |
 <com:TLinkButton Text="Supprimer"
  OnClick="deletePost"
  Attributes.onclick="javascript:if(!confirm('Etes vous sûr ?')) return false;" />
</com:TControl>

<p>
Auteur:
<com:TLiteral Text="<%= $this->Post->author->username %>" /><br/>
Heure:
<com:TLiteral Text="<%= date('m/d/Y h:m:sa', $this->Post->create_time) %>" />
</p>

<p>
<com:TLiteral Text="<%= $this->Post->content %>" />
</p>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
</com:TContent>
```

Plusieurs expressions PHP sont utilisées dans le gabarit. L'expression *$this->Post* fait référence à la propriété définie dans la classe de *ReadPost*. Elle représente l'objet *PostRecord* correspondant au message actuel.

Information: Même si nous utilisons régulièrement des expressions dans nos gabarits, nous n'en abusons pas. Une des règles principales pour savoir si l'on ***doit utiliser une expression est l'expression doit être une propriété ou une simple mise en forme d'une propriété****. En suivant cette ligne de conduite, nous nous assurons d'une bonne séparation entre le contenu et la présentation, sans perdre en flexibilité.*

Nous pouvons aussi remarquer dans le gabarit précédent, que, nos deux boutons sont entourés d'un *TControl* dont la propriété 'visible' est déterminée par l'expression *$this->canEdit()*. Pour le bouton "Supprimer", nous utilisons une boite de dialogue javascript pour confirmer la suppression du message.

Information: Tous les contrôles PRADO, ont une propriété très utile Attributes qui accepte n'importe quelle paire de valeurs (nom-valeur). La plupart des contrôles PRADO répercutent directement ces informations dans la balise HTML. Par exemple, dans le bouton "Supprimer" nous définissons onclick qui est directement reporté dans la balise <a> sous forme d'un attribut onclick.

Création du fichier de classe

Dans le gabarit précédent, nous voyons que notre classe doit implémenter le gestionnaire d'évènement *deletePost()* (attaché à l'évènement *Onclick* de notre bouton "Supprimer"). Nous devons aussi lire les données du message dont l'ID est passé par un paramètre GET.

Information: Nous implémentons la fonctionnalité suppression dans le classe ReadPost parce qu'il est classique de faire ainsi. Quand l'utilisateur clique sur le bouton "Supprimer", une boite de dialogue demande confirmation de la suppression. Si l'utilisateur confirme, l'évènement OnClick du bouton "Supprimer" est déclenché.

```
class ReadPost extends TPage
{
 private $_post;
 /**
  * lis les données du message.
  * cette méthode est appelée lors de l'initialisation de la page
  * @param mixed param : paramètres de l'évènement
  */
 public function onInit($param)
 {
  parent::onInit($param);
  // id du message passé par un paramètre GET
  $postID=(int)$this->Request['id'];
  // lis le message ainsi que les données correspondantes à l'auteur
  $this->_post=PostRecord::finder()->withAuthor()->findByPk($postID);
  if($this->_post===null)  // si l'id du message est invalide
   throw new THttpException(500, 'Impossible de trouver le message demandé.');
  // défini le titre de la page comme étant celui du message
  $this->Title=$this->_post->title;
 }

 /**
  * @return PostRecord retourne l'objet PostRecord correspondant au message
  */
 public function getPost()
 {
  return $this->_post;
 }
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
	/**
	 * supprime le message actuellement visualisé
	 * cette méthode est appelée par l'évènement OnClick du bouton "Supprimer"
	 */
	public function deletePost($sender,$param)
	{
		// seul l'auteur ou un administrateur peuvent supprimer le message
		if(!$this->canEdit())
			throw new THttpException("Nous n'êtes pas autorisé à effectuer cette action.");
		// le supprime de la base de données
		$this->_post->delete();
		// redirige le navigateur vers la page d'accueil
		$this->Response->redirect($this->Service->DefaultPageUrl);
	}

	/**
	 * @return boolean infiquant si le message peut être modifier ou supprimer par l'utilisateur actuel
	 */
	public function canEdit()
	{
		// seul l'auteur ou un administrateur peuvent modifier/supprimer le message
		return $this->User->Name===$this->Post->author_id || $this->User->IsAdmin;
	}
}
```

Test

Pour tester notre page *ReadPost*, allons à l'URL *http://hostname/blog/index.php* et cliquons sur le titre du seul message affiché. Notre navigateur devrait afficher le résultat suivant avec l'URL *http://hostname/blog/index.php?page=ReadPost&id=1*. Notez que si vous n'êtes pas connecté, les deux boutons ne sont pas visibles.

My PRADO Blog

first post

Edit | Delete

Author: admin
Time: 04/04/2007 01:04:22pm

this is my first post

Logout

V-D - Création de la page nouveau message 'NewPost'

La page *NewPost* permet aux utilisateurs authentifiés de créer des nouveaux messages. Elle doit afficher un formulaire permettant la saisie des informations du message.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Parce que la page *NewPost* ne peut être vu que par les utilisateurs authentifiés, nous ajoutons un fichier de configuration de page *config.xml* dans le dossier *protected/pages/posts*. Cette configuration indique que les invités ne peuvent voir les pages *NewPost* et *EditPost* qui sera implémentée dans la section suivante.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <authorization>
    <deny pages="NewPost,EditPost" roles="?" />
  </authorization>
</configuration>
```

Vu le nombre grandissant de pages, nous allons modifier le pied de page de notre gabarit principal pour qu'il inclus des liens vers : la page d'accueil, la page nouvel utilisateur *NewUser* (visible seulement par les administrateurs), et la page à venir : nouveau message *NewPost* (visible seulement par les utilisateurs authentifiés).

```
<div id="footer">
<com:THyperLink Text="Accueil"
 NavigateUrl="<%= $this->Service->DefaultPageUrl %>" />

<com:THyperLink Text="Nouveau message"
 NavigateUrl="<%= $this->Service->constructUrl('posts.NewPost') %>"
 Visible="<%= !$this->User->IsGuest %>" />

<com:THyperLink Text="Nouvel utilisateur"
 NavigateUrl="<%= $this->Service->constructUrl('users.NewUser') %>"
 Visible="<%= $this->User->IsAdmin %>" />
...autres liens...
</div>
```

Nous allons maintenant créer deux fichiers *protected/pages/posts/NewPost.page* et *protected/pages/posts/NewPost.php* contenant respectivement le gabarit et la classe de notre page.

Création du gabarit

Le gabarit de *NewPost* contient une référence à un **TTextBox** pour saisir le titre de notre message et à un **THtmlArea** pour saisir le contenu. Ce dernier est un éditeur WYSIWYG HTML. Pour contrôler les valeurs saisies, nous associons des validateurs aux contrôles précédents.

```
<%@ Title="Mon Blog - Nouveau Message" %>

<com:TContent ID="Main">

<h1>Création nouveau message</h1>

<span>Titre:</span>
<com:TRequiredFieldValidator
 ControlToValidate="TitleEdit"
 ErrorMessage="Veuillez indiquer un titre."
 Display="Dynamic" />
<br/>
<com:TTextBox ID="TitleEdit" Columns="50" />

<br/>
<span>Message:</span>
<com:TRequiredFieldValidator
 ControlToValidate="ContentEdit"
 ErrorMessage="Veuillez indiquer le contenu du message."
 Display="Dynamic" />
<br/>
<com:THtmlArea ID="ContentEdit" />

<br/>
<com:TButton Text="Ajouter" OnClick="createButtonClicked" />

</com:TContent>
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

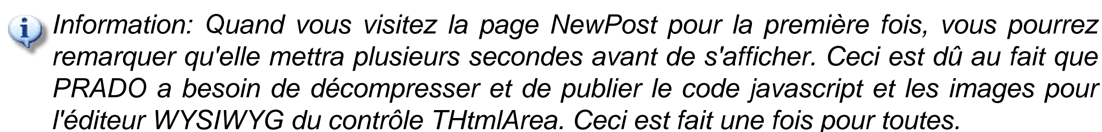
Création du fichier de classe

Dans le gabarit précédent, nous voyons que la fonction principale de notre page est l'appel de la méthode *createButtonClicked()* implémenté par un évènement *OnClick* attaché au bouton "Ajouter".

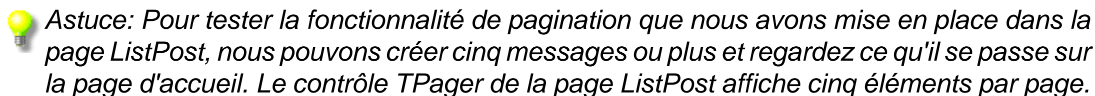
```
class NewPost extends TPage
{
 /**
  * création d'un nouveau message si toutes les données sont valides.
  * cette méthode est appelée par l'évènement OnClick du bouton "Ajouter".
  * @param mixed sender : celui qui a généré l'évènement
  * @param mixed param : paramètres de l'évènement
  */
 public function createButtonClicked($sender,$param)
 {
  if($this->IsValid)  // tous les validateurs sont Ok ?
  {
   // créer un nouvel objet PostRecord avec les données du formulaire
   $postRecord=new PostRecord;
   // utiliser SafeText à la place de Text évite les attaques XSS
   $postRecord->title=$this->TitleEdit->SafeText;
   $postRecord->content=$this->ContentEdit->SafeText;
   $postRecord->author_id=$this->User->Name;
   $postRecord->create_time=time();
   $postRecord->status=0;

   // enregistre les données dans la BDD par la méthode save de l'Active Record
   $postRecord->save();

   // redirige le navigateur vers le message nouvellement créé
   $url=$this->Service->constructUrl('posts.ReadPost',array('id'=>$postRecord->post_id));
   $this->Response->redirect($url);
  }
 }
}
```

Test

Pour tester notre page *NewPost*, identifiez-vous auparavant et cliquez sur le lien *Nouveau message* dans le pied de page. Le navigateur affiche le résultat suivant avec comme URL *http://hostname/blog/index.php?page=NewPost*.



Information: Quand vous visitez la page NewPost pour la première fois, vous pourrez remarquer qu'elle mettra plusieurs secondes avant de s'afficher. Ceci est dû au fait que PRADO a besoin de décompresser et de publier le code javascript et les images pour l'éditeur WYSIWYG du contrôle THtmlArea. Ceci est fait une fois pour toutes.



Astuce: Pour tester la fonctionnalité de pagination que nous avons mise en place dans la page ListPost, nous pouvons créer cinq messages ou plus et regardez ce qu'il se passe sur la page d'accueil. Le contrôle TPager de la page ListPost affiche cinq éléments par page.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

Create New Post

Title:

Content:

-- Format -- -- Font family -- -- Font size --

HTML

Path:

Create

Home New Post New User Logout

prado powered

V-E - Création de la page modification d'un message 'EditPost'

La page *EditPost* est fournie aux auteurs et administrateurs pour modifier les messages. Comme la page *NewPost*, elle affiche un formulaire permettant de modifier les données d'un message.

Nous créons deux fichiers *protected/pages/posts/EditPost.page* et *protected/pages/posts/EditPost.php* contenant respectivement le gabarit et la classe de notre page.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création du gabarit

Le gabarit de la page *EditPost* est très proche de celui de la page *NewPost* template. Seul le titre et le texte du bouton sont différents.

```
<%@ Title="Mon Blog - Modification Message" %>

<com:TContent ID="Main">

<h1>Modification message</h1>

<span>Titre:</span>
<com:TRequiredFieldValidator
 ControlToValidate="TitleEdit"
 ErrorMessage="Veuillez indiquer un titre."
 Display="Dynamic" />
<br/>
<com:TTextBox ID="TitleEdit" Columns="50" />

<br/>
<span>Message:</span>
<com:TRequiredFieldValidator
 ControlToValidate="ContentEdit"
 ErrorMessage="Veuillez indiquer le contenu du message."
 Display="Dynamic" />
<br/>
<com:THtmlArea ID="ContentEdit" />

<br/>
<com:TButton Text="Enregistrer" OnClick="saveButtonClicked" />

</com:TContent>
```

Création du fichier de classe

La classe de *EditPost* est un peu plus complexe que celle de la page *NewPost* parce qu'elle doit lire les informations auparavant. Elle doit aussi vérifier les autorisations. En particulier, elle doit s'assurer que le message ne puisse être modifié que par l'auteur ou par un administrateur. Ces vérifications d'autorisation ne sont pas fournies par PRADO.

```
class EditPost extends TPage
{
 /**
  * initialise les contrôles de saisies avec les données du message.
  * cette méthode est appelée lors de l'initialisation de la page
  * @param mixed param : paramètres de l'évènement
  */
 public function onInit($param)
 {
  parent::onInit($param);
  // récupère les données de l'utilisateur. Equivalent à:
  // $postRecord=$this->getPost();
  $postRecord=$this->Post;
  // vérification des droits: seul l'auteur ou un administrateur peuvent modifier le message
  if($postRecord->author_id!==$this->User->Name && !$this->User->IsAdmin)
   throw new THttpException(500,'Vous n êtes pas autoriser à modifier ce message.');

  if(!$this->IsPostBack)  // est-ce le premier appel à la page
  {
   // rempli les contrôles avec les données du message
   $this->TitleEdit->Text=$postRecord->title;
   $this->ContentEdit->Text=$postRecord->content;
  }
 }

 /**
  * Enregistre si toutes les validations sont Ok
  * cette méthode répond à l'évènement OnClick du bouton "Enregistrer".
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
  * @param mixed sender : celui qui a généré l'évènement
  * @param mixed param : paramètres de l'évènement
  */
 public function saveButtonClicked($sender,$param)
 {
  if($this->IsValid)  // toutes les validations sont ok ?
  {
   // récupère les données de l'utilisateur. Equivalent à:
   // $postRecord=$this->getPost();
   $postRecord=$this->Post;

   // affecte les données saisies aux champs de la BDD
   $postRecord->title=$this->TitleEdit->SafeText;
   $postRecord->content=$this->ContentEdit->SafeText;

   // enregistre les données par la méthode save de l'Active Record
   $postRecord->save();

   // redirige le navigateur vers la page ReadPost
   $url=$this->Service->constructUrl('posts.ReadPost',array('id'=>$postRecord->post_id));
   $this->Response->redirect($url);
  }
 }

 /**
  * retourne les données du message devant être modifiées.
  * @return PostRecord les données devant être modifiés.
  * @throws THttpException si le message est inexistant.
  */
 protected function getPost()
 {
  // l'ID du message devant être modifié passé par un paramètre GET
  $postID=(int)$this->Request['id'];
  // utilise Active Record pour lire le message correspondant à cet ID
  $postRecord=PostRecord::finder()->findByPk($postID);
  if($postRecord===null)
   throw new THttpException(500,'Message inexistant.');
  return $postRecord;
 }
}
```

Test

Pour tester notre page *EditPost*, authentifiez-vous auparavant et allez à l'URL *http://hostname/blog/index.php?page=EditPost&id=1*. Cette URL peut aussi être atteinte par le bouton "Modifier" de notre page de détail.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

Edit Post

Title:

first post

Content:

-- Format -- | -- Font family -- | -- Font size -- | B I U ABC x₂ x² | HTML

this is my first post

Path:

Save

Home New Post New User Logout

VI - 5ième jour: Refactorisation et déploiement

VI-A - Utilisation des Thèmes et des Skins

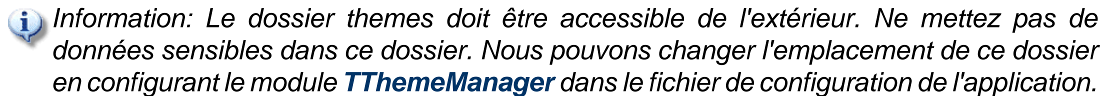
PRADO propose un support intrinsèque des **thèmes**. En utilisant les thèmes, nous pouvons mieux séparer la logique applicative de la présentation et nous pouvons aussi changer facilement la présentation générale de notre blog.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Création des thèmes

Nous devons auparavant créer un dossier *themes*. C'est le dossier parent de tous les thèmes pour une application de PRADO. Chaque sous-dossier devient ainsi un thème dont le nom est le nom du sous-dossier.

Pour créer un thème nommé *Basic*, nous créons un sous-dossier *theme/Basic*. Dans ce dossier, nous pouvons mettre des feuilles de styles dépendantes du thème, des fichiers Javascript, des images et des fichiers skins.


*Information: Le dossier themes doit être accessible de l'extérieur. Ne mettez pas de données sensibles dans ce dossier. Nous pouvons changer l'emplacement de ce dossier en configurant le module **TThemeManager** dans le fichier de configuration de l'application.*

Création de la feuille de style

Dans le dossier *themes/Basic*, nous créons un fichier CSS nommé *style.css*. Quand une page utilise ce thème, PRADO importe automatiquement la feuille de style dans cette page. Le même traitement est appliqué aux fichiers Javascript.

Le contenu du fichier CSS est le suivant :

```
body {
 font-family: verdana, 'trebuchet ms', sans-serif;
 font-size: 10pt;
 background: white;
}
#page {
 margin: 0 auto 0 auto;
 width: 600px;
}
#footer {
 text-align: center;
 margin-top: 10px;
 padding: 10px;
 border-top: 1px solid silver;
}
.post-box {
 margin-bottom: 10px;
 padding: 5px;
}
.post-box h3 {
 padding: 5px;
 font-size: 13pt;
 background: lightgray;
}
.post-box a {
 color: black;
 text-decoration: none;
}
.post-box a:hover {
 color: red;
}
```

Création du fichier de Skin

Nous utilisons des *skin* pour initialiser les propriétés des contrôles PRADO. Les fichiers *skin* sont enregistrés avec une extension *.skin* dans le dossier du thème. Chaque fichier *skin* peut contenir plusieurs modèles pour un ou plusieurs types de contrôles.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Pour notre test, nous allons créer un fichier *skin* qui changera la couleur de fond de nos liens dans le pied de page. Nous créons un fichier nommé *button.skin* dans le dossier du thème *themes/Basic*.

```
<com:THyperLink SkinID="MainMenu" BackColor="lightgreen" />
```

Le fichier *button.skin* contient une seule définition pour les contrôles de type *THyperLink* dont la propriété *SkinID* est *MainMenu*. La définition applique une couleur vert-clair comme couleur de fond du contrôle.

En accord avec cette définition, nous modifions notre fichier *protected/common/MainLayout.tpl* pour appliquer aux liens du pied de page la valeur *MainMenu* à la propriété *SkinID*.

```
......
<div id="footer">
......
<com:THyperLink Text="Home" SkinID="MainMenu"
 NavigateUrl="<%= $this->Service->DefaultPageUrl %>" />

<com:THyperLink Text="New Post" SkinID="MainMenu"
 NavigateUrl="<%= $this->Service->constructUrl('posts.NewPost') %>"
 Visible="<%= !$this->User->IsGuest %>" />
......
</div>
......
```

Information: La syntaxe des fichiers skin est très proche de celle des gabarits. Chaque balise <com:> définit la présentation d'un type de contrôle. PRADO concatène automatiquement les fichiers skin pour un thème et applique le tout lorsque la page est affichée.

Utilisation du thème

Pour utiliser le thème que nous venons juste de créer, nous modifions notre fichier de configuration de l'application comme ci-après. Comme vous pouvez le voir, nous affectons la valeur *Basic* (le nom du thème) à la priorité *Theme* pour toutes les pages.

```
......
  <services>
    <service id="page" class="TPageService" DefaultPage="posts.ListPost">
      <pages MasterClass="Application.layouts.MainLayout" Theme="Basic" />
    </service>
  </services>
......
```

Information: Il est possible de préciser différents thèmes pour différentes pages, et ceci peut-être faits soit en modifiant le fichier de configuration de page soit par programmation (propriété Theme). En dernier recours, on peut le faire dans la méthode onPreInit() de la page, ceci parce que PRADO applique le thème au début du cycle de vie de la page.

Test

Pour voir la nouvelle présentation de notre site, allons à l'URL *http://hostname/blog/index.php*. Nous pouvons constater que la mise en page, les polices, les bordures sont modifiées. De même, la couleur de fond des liens en pied de page est vert-clair.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

first post

Author: admin
Time: 04/04/2007 01:04:22pm

this is my first post

Home Login

VI-B - Gestion et journalisation d'erreur

Si vous tentez de naviguez à l'URL *http://hostname/blog/index.php?page=EditPost&id=100*, vous verrez la page d'erreur suivante parce que le message avec l'ID 100 n'existe pas pour le moment. Nous voudrions personnaliser cette page d'erreur de manière à ce qu'elle garde la présentation générale du site. Nous voudrions aussi journaliser cette erreur pour étudier le comportement des utilisateurs. Dans cette section, nous allons mettre en place ces deux fonctionnalités.

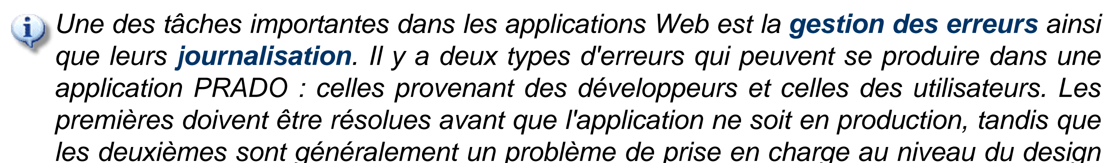
Internal Server Error

Unable to find the specified post.

An internal error occurred while the Web server was handling your request. Please

Thank you.

2007-06-29 08:35 Apache/2.0.59 (Win32) PHP/5.2.0 SVN/1.4.3 DAV/2 PRADO/3.1.0



Une des tâches importantes dans les applications Web est la ***gestion des erreurs*** *ainsi que leurs* ***journalisation****. Il y a deux types d'erreurs qui peuvent se produire dans une application PRADO : celles provenant des développeurs et celles des utilisateurs. Les premières doivent être résolues avant que l'application ne soit en production, tandis que les deuxièmes sont généralement un problème de prise en charge au niveau du design*

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

et doivent être gérées proprement (ie: journaliser cette erreur et indiquer à l'utilisateur que faire après). PRADO fournit un ensemble de fonctionnalités très utiles pour gérer et journaliser les erreurs.

Personnalisation de la gestion d'erreur

PRADO charge de manière implicite un module de gestion d'erreurs. Nous voulons personnaliser ce module pour qu'il affiche une page spécifique pour les erreurs causées par les utilisateurs. Nous modifions donc notre application comme ci-dessous :

```
......
<modules>
    ......
    <module class="Application.BlogErrorHandler" />
    ......
</modules>
......
```

La classe *BlogErrorHandler* comme spécifiée ci-dessus est un nouveau gestionnaire d'erreur que nous allons créer après. Il étend et remplace le module par défaut *TErrorHandler*.

Nous créons un fichier nommé *protected/BlogErrorHandler.php* avec le contenu suivant. La classe *BlogErrorHandler* surcharge deux méthodes de *TErrorHandler* :

- *getErrorTemplate()* - cette méthode renvoie le gabarit utilisé pour afficher un message d'erreur utilisateur.
- *handleExternalError()* - cette méthode est appelée lorsqu'une erreur utilisateur se produit et elle affiche l'erreur.

```
Prado::using('System.Exceptions.TErrorHandler');
Prado::using('Application.BlogException');

class BlogErrorHandler extends TErrorHandler
{
 /**
  * Renvoi le fichier gabarit utilisé pour afficher l'erreur.
  * Cette méthode surcharge la méthode originale.
  */
 protected function getErrorTemplate($statusCode,$exception)
 {
  // on utilise notre propre gabarit pour BlogException
  if($exception instanceof BlogException)
  {
   // récupère le chemin du fichier de gabarit : protected/error.html
   $templateFile=Prado::getPathOfNamespace('Application.error','.html');
   return file_get_contents($templateFile);
  }
  else // sinon on utilise le gabarit par défaut.
   return parent::getErrorTemplate($statusCode,$exception);
 }

 /**
  * Gère les erreurs causées par les utilisateurs.
  * Cette méthode surcharge la méthode originale.
  * Elle est appelée lorsqu'une exception utilisateur est générée.
  */
 protected function handleExternalError($statusCode,$exception)
 {
  // Journaliser l'erreur (seulement pour BlogException)
  if($exception instanceof BlogException)
   Prado::log($exception->getErrorMessage(),TLogger::ERROR,'BlogApplication');
  // appelle l'implémentation de la classe parente
  parent::handleExternalError($statusCode,$exception);
 }
}
```


Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Dans le code précédent, nous spécifions que lorsqu'une exception de type *BlogException* est générée, nous utilisons le gabarit *protected/error.html* pour afficher l'erreur. Par ailleurs, nous devons créer la classe *BlogException* et remplacer toutes les occurrences de *THttpException* dans notre code (par exemple dans les pages 'Mise à jour des utilisateurs' et 'Détail d'un message'). Nous devons aussi créer le gabarit *protected/error.html*. La classe *BlogException* hérite de *THttpException* et est vide. Le fichier de classe est enregistré sous *protected/BlogException.php*.

```
class BlogException extends THttpException
{
}
```

Ci-dessous le contenu du gabarit *protected/error.html*. Remarquez que ce gabarit n'est pas un gabarit PRADO, ceci parce qu'il ne reconnait qu'un nombre limité de mots clés, par exemple *%%ErrorMessage%%*, *%%ServerAdmin%%*.

```
<html>
<head>
<title>%%ErrorMessage%%</title>
</head>
<body>
<div id="page">
<div id="header">
<h1>Mon Blog</h1>
</div>
<div id="main">
<p style="color:red">%%ErrorMessage%%</p>
<p>
Une erreur est apparue lors du traitement de votre demande.
</p>
<p>
Si vous pensez que c'est une erreur de notre serveur, veuillez contacter <a href="mailto:%%ServerAdmin%
%">webmaster</a>.
</p>
</div>
</body>
</html>
```

Journalisation des erreurs

Dans la méthode *handleExternalError()* de *BlogErrorHandler*, nous appelons *Prado::log()* pour journaliser l'erreur si elle est de type *BlogException*. L'erreur est stockée en mémoire. Pour enregistrer le journal d'erreur sur un support non volatil, tel que le disque dur ou une base de données, nous devons indiquer à PRADO comment procéder. Ceci est fait par la configuration d'application suivante :

```
......
<modules>
    ......
    <module id="log" class="System.Util.TLogRouter">
        <route class="TFileLogRoute" Categories="BlogApplication" />
    </module>
    ......
</modules>
......
```

Dans le code ci-dessus, nous ajoutons une "route" pour enregistrer le journal d'erreur dans un fichier. Nous spécifions aussi le filtre de catégorie *BlogApplication*, de manière à ce que seules les erreurs de type *BlogApplication* soient sauvegardées. Cette possibilité permet de réduire la taille du journal et d'en améliorer la lisibilité.

Test

Pour voir comme notre blog se comporte suite à une demande invalide, nous naviguons à l'URL *http://hostname/blog/index.php?page=posts.ReadPost&id=100*. Nous devrions voir la page suivante qui est différente de celle vue précédemment.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

My PRADO Blog

Unable to find the specified post.

The above error happened when the server was processing your request.

If you think this is a server error, please contact the webmaster.

Si nous regardons dans le dossier *protected/runtime*, nous devrions y trouver un fichier nommé *prado.log*. C'est le journal d'erreur que nous venons juste de paramétrer. Le fichier pourrait contenir quelque chose comme :

```
Jun 28 22:15:27 [Error] [BlogApplication] Unable to find the specified post.
Jun 29 08:42:57 [Error] [BlogApplication] Unable to find the specified post.
```

VI-C - Amélioration des performances

Avant le déploiement de notre blog, nous voudrions améliorer les performances.

Changer le mode de fonctionnement de l'application

Une application PRADO peut-être configurée pour fonctionner suivant différents modes. Par défaut, elle fonctionne en mode *Debug*, mode qui génère beaucoup de message de journalisation et qui, en cas d'erreurs, affiche la pile des appels et l'emplacement de l'erreur. Ce comportement est préférable en cours de développement, mais pas en production. Pour changer le mode de fonctionnement de *Debug* à *Normal* (qui signifie "en production"), nous devons modifier le fichier de configuration de l'application comme ci-dessous :

```
<?xml version="1.0" encoding="utf-8"?>
<application id="blog" mode="Normal">
    ......
</application>
```

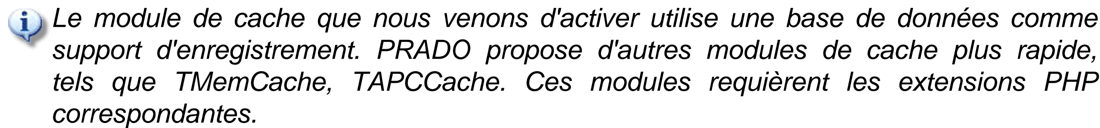
Activer le cache

Beaucoup de travail d'analyse est effectué par une application PRADO : fichier XML de configuration, thèmes, skins, etc. Pour chaque requête utilisateur, PRADO doit refaire l'analyse. Pour éviter ce travail, nous allons activer le *cache*. Pour ce faire, modifions notre fichier de configuration de l'application comme ci-dessous :

```
......
<modules>
    ......
    <module id="cache" class="System.Caching.TDbCache" />
    ......
</modules>
......
```

Maintenant, après avoir requis n'importe quelle page de notre blog, nous devrions trouver un fichier nommé *sqlite3.cache*. C'est un fichier de base de données qui mémorise les éléments analysés : gabarits, configurations, etc.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



Le module de cache que nous venons d'activer utilise une base de données comme support d'enregistrement. PRADO propose d'autres modules de cache plus rapide, tels que TMemCache, TAPCCache. Ces modules requièrent les extensions PHP correspondantes.

Utilisation de pradolite.php

Afficher une page PRADO requiert des dizaines de fichiers PHP, ce qui est une cause de perte de temps. Ces fichiers comportent aussi beaucoup de commentaires qui permettent de générer la documentation des API. Dans le but de réduire ce coût, nous modifions notre fichier *index.php* et remplaçons *prado.php* par *pradolite.php*. Ce dernier est un gros fichier incluant les fichiers PHP nécessaires et dont on a retiré les commentaires.

Autres techniques

D'autres techniques sont disponibles pour améliorer les performances d'une application PRADO. D'après notre expérience, un des goulets d'étranglement dans une application Web, est l'accès aux bases de données. Les requêtes en base de données prennent souvent du temps, ce qui dégrade le temps d'affichage d'une page. Le *cache* est la principale solution à ce problème. Le module de *cache* activé dans notre fichier de configuration d'application peut aussi être utilisé dans ce but.

Pour une page relativement stable et souvent consultée, le **cache de sortie** doit être envisagé. Le *cache de sortie* met en *cache* les parties sélectionnées d'une page. Ceci peut améliorer les performances des pages mises en *cache* de manière significative.

Il a été prouvé que les solutions de *cache* côté serveur étaient très efficaces pour améliorer les performances d'une application PRADO. Par exemple, nous avons observé qu'en utilisant le *Zend Optimizer*, le RPS (requêtes par seconde) peut être multiplié par 10. Bien sûr, ceci au risque d'avoir des pages périmées, tandis que les solutions de *cache* de PRADO garantissent la validité des pages fournies.

VI-D - Résumé

Nous pouvons finalement déployer notre blog. Pour cela, nous devons juste copier le dossier *blog* complet vers le dossier du serveur Web. Nous pourrions avoir besoin de modifier *index.php* pour qu'il puisse trouver le chemin vers l'emplacement où a été installé le framework PRADO.

Nous avons donc fini notre blog. Le processus peut paraitre complexe vu que nous avons passé pas loin de cinq jours pour y arriver. Toutefois, comme nous l'avions dit au début, le but de ce tutoriel est d'aider les développeurs PRADO à appréhender les principales techniques de PRADO.Le tutoriel n'avais pas pour but de finir un blog en cinq minutes, sinon nous n'aurions rien appris.

En résumé, développer une application de gestion de base de données PRADO nécessite les étapes suivantes :

1. Analyse et création de la base de données
2. Créer le squelette de l'application avec *prado-cli*
3. Mise en place de la gestion d'erreur pour gérer les erreurs d'utilisations
4. Création et mise en place du thème
5. Création et mise en place des gabarits principaux
6. Création de la connexion et des classes d'accès aux données
7. Création des différentes pages
8. Test et amélioration des performances
9. Déploiement

Contrairement à l'ordre de notre tutoriel, la gestion d'erreur et la création des thèmes sont placées au début du processus. Ceci est dû au fait que des changements d'ordre généraux sont la plupart du temps requis. Par exemple,

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

nous avons dû remplacer *THttpException* par *BlogException* dans notre tutoriel. Si vous définissez vos feuilles de styles plus tôt, vous pourrez plus facilement les utiliser au cours de la création des gabarits de pages.

Un dernier conseil, essayez de penser orienté objet pendant la phase d'analyse et d'implémentation. Utilisez l'héritage le plus souvent, et vous trouverez que le projet est plus facile à développer en équipe. Il vous sera aussi plus facile de réutiliser votre code et ainsi vos futurs projets seront finis plus rapidement.

Les sources présentées sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2007 - pradosoft.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.