Travaux Pratiques - Play Framework 2

Démonstration de l'application

Vous pourrez trouver la version déployée de l'application à l'adresse suivante :

```
celestialbeing.cloudapp.net
```

Configuration du projet

Pour faciliter le développement, on va créer un alias vers le script *activator*. Tous les projets Play Framework pourront ainsi être créés et lancés depuis n'importe quel endroit du PC. Effectuez ces trois étapes :

```
vi ~/.bashrc
alias play='/gfs/votre_identifiant_fil/activator-1.2.10-minimal/activator'
source ~/.bashrc
```

Importez le projet à partir de GitHub dans le dossier */gfs/votre_identifiant_fil* :

```
git clone https://github.com/JimiPepper/tuto-play-framework-iagl.git
```

Toutes les étapes qui suivront s'effectueront à partir du répertoire téléchargé sur Git

Association d'une base de données MySQL

Vous allez modifier les paramètres pour permettre à l'application d'accéder à une base de données MySQL. Pour cela, ouvrez le fichier de configuration de l'application *application.conf* situé dans le dossier *conf* et modifiez :

1. l'url (*db.default.url*) de la base de données
2. votre identifiant (*db.default.user*) et votre mot de passe (*db.default.password*) afin de pouvoir vous connecter

Note :

Votre identifiant : votre nom
Votre mot de passe : play
Url de la BDD : jdbc:mysql//bl0p20/Votre_Identifiant
Vous pouvez vous aider de l'url suivante

Importation du projet

Avant d'importer le projet sur votre IDE préféré, il faut le convertir. Pour cela, placez-vous dans le répertoire du projet cloné et tapez l'une des commandes suivantes

- pour Eclipse : `play eclipse`
- pour IntelliJ : `play idea`

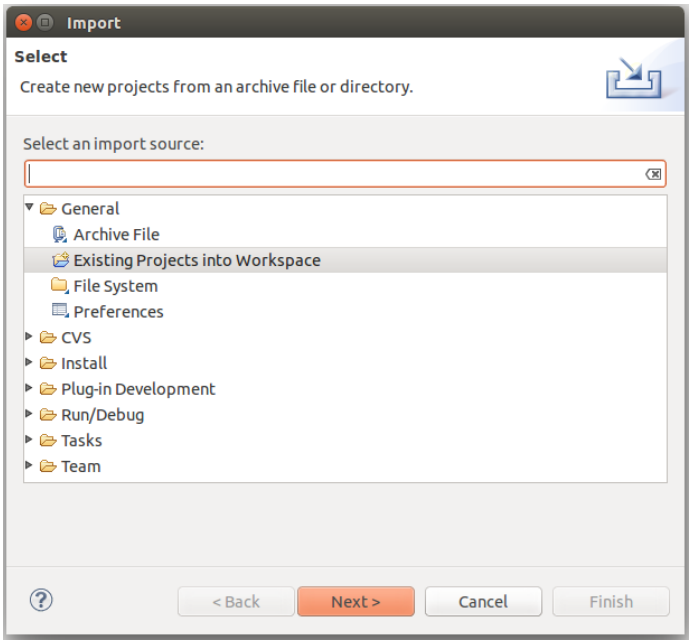
Pour configurer votre projet sur un autre IDE, veuillez consulter la documentation suivante

NB : La suite a été réalisée avec Eclipse, nous ne savons pas comment cela va se comporter avec IntelliJ. À vous de voir !

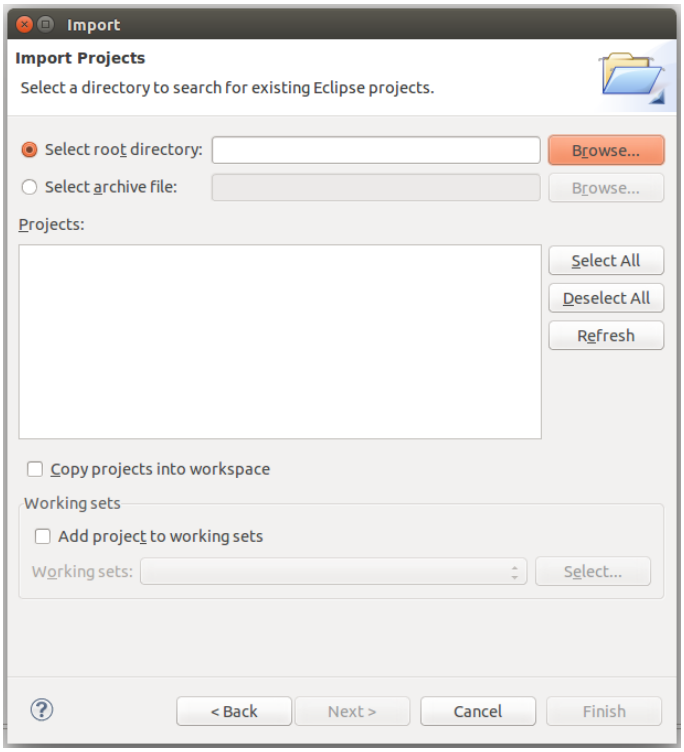
Vous avez maintenant la possibilité d'importer votre projet dans Eclipse :

Menu : File -> Import

Une fenêtre Import s'ouvre. Sélectionnez "Existing Projects Into Workspace"



Sélectionnez le répertoire de votre projet



Lancement du projet

Afin d'accéder à l'application, il faut lancer le serveur. Ainsi, tapez la commande suivante : `play run` . L'application doit être disponible à l'adresse : localhost:9000

Il vous sera demandé d'exécuter un script afin de modifier la base de données :

- la première fois où vous lancerez à l'application
- à chaque modification sur les modèles

Exercice 1 : Ajout de la page contact

But: Dans cet exercice, vous allez devoir créer une page de contact vous permettant d'envoyer un mail aux administrateurs.

Nouvelles notions abordées :
- **Création d'une route**
- **Modification d'un Controller**

La vue *contact.scala.html* permettant d'envoyer un mail existe déjà. Vous la trouverez dans le dossier *app/views*. Pour y accéder, nous avons besoin que le Controller *Contact* renvoie cette vue à l'aide de la méthode *contact()* quand l'url http://localhost:9000/contact est appelée.

Pour commencer, il faut créer une route afin de pouvoir accéder à la page contact.

1. ouvrez le fichier "*conf/routes*".
2. rajoutez une route "*/contact*" en vous aidant des routes déjà créées.

Maintenant, vous allez créer le lien "Contact" dans le menu. Le menu est défini dans la vue *main.scala.html*. Vous pouvez vous aider des liens déjà écrits dans cette vue.

En cas de succès, lorsque vous cliquez sur le lien "Contact", le formulaire de contact doit s'afficher.

Puis dans un second temps, il faudra transmettre à ce Controller les données envoyées par ce formulaire pour envoyer un mail aux administrateurs en modifiant la méthode *sendMail* du Controller *Contact*. Pour cela, il faut :

1. Associer l'action à réaliser (ici *sendMail*) au formulaire
2. Récupérer les données fournies par le formulaire de contact
3. Construire un mail
4. Envoyer ce mail.
5. Rediriger vers la page de contact et non plus vers la page d'accueil suite à l'envoi d'un mail

Nous utiliserons le serveur SMTP de la faculté pour envoyer des mails, il est nécessaire d'activer une connexion authentifiée en utilisant les méthodes : *setStartTLSEnabled(boolean activate)* et *setStartTLSRequired(boolean activate)* au lieu *setSSLOnConnect(boolean activate)*.

Il vous reste une dernière étape de configuration avant de pouvoir envoyer des mails. En effet, vous devez ajouter dans la section "*Email Configuration*" du fichier application.conf situé dans le répertoire conf :

1. votre identifiant de la faculté (identifiant pour vous connecter à votre boite mail)
2. votre mot de passe
3. l'adresse mail du destinataire de votre mail

Afin de vérifier si votre mail a bien été envoyé, il vous suffira de regarder si vous avez bien reçu le mail sur l'adresse e-mail indiquée dans la partie précédente.

Liens à consulter :

Ajouter une route : voir la section The routes file syntax

Créer un mail et envoyer ce mail : consulter la section **Simple mail**

Exercice 2 Authentification

But : Vous allez mettre en place une solution vous permettant de vous connecter en tant qu'administrateur afin de pouvoir rajouter des images par la suite.

Nouvelles notions abordées :
- **Création d'une vue**
- **Création d'un Controller**
- **Utilisation des Ebean**
- **Utilisation des sessions**

Premièrement, vous allez devoir écrire la vue *loginAdmin.scala.html*. Celle-ci doit contenir un formulaire avec :
- un champ pour le login
- un champ pour le mot de passe
- un bouton submit

Attention, n'oubliez pas de :
1. Créer dans un nouveau controller *Login*, la méthode index qui permettra de renvoyer cette vue quand l'url "*admin/login*" est appelée.
2. Ajouter la route afin d'accéder à ce formulaire à partir du lien "*Partie Administrateur*" dans le menu.
3. Créer le lien "*Partie Administrateur*" dans le menu.

Vous devez maintenant être capable d'afficher la page de contact en appuyant sur le lien "Partie Administrateur".

Pour la suite, il faut traiter le formulaire à l'aide de la méthode *connect* dans le controller *Login* . Vous devez alors réaliser ces différentes étapes :
1. Récupérer le formulaire.
2. Récupérer les informations saisies dans le formulaire afin de vérifier si celles-ci sont bien en relation avec un administrateur (s'aider de Ebean).
3. Créer une session si l'étape précédente est vérifiée (grâce à *AdminToken.java* qui vous est fourni. Il faut aussi décommenter la méthode).

4. Rediriger vers la page d'administration (*/admin/image/list*) si les identifiants sont corrects. Dans le cas contraire, rediriger vers de connexion.

Voici le pseudo code de la méthode connect :

```
/**
* Connects user if it is an administrator
*/
public static Result connect() {
   /* GET FORM */

   /* LOOK FOR USER IN SGBD */

   /* CREATE AUTHENTIFICATION SESSION */

   /* REDIRECT */
}
```

Il vous faut maintenant associer une action au formulaire que vous avez créé. Pour cela :

1. Ajouter la route */admin/connect* (avec le mot Http **POST**). Cette route doit pouvoir appeler la méthode *connect* du controller *Login*.
2. Associer l'action *connect* au formulaire dans la vue *loginAdmin*. Vous pouvez vous aider du formulaire d'envoi de mail.

Vous devez normalement pouvoir vous connecter à l'aide de l'identifiant jiji et du mot de passe misa.

Pour améliorer le comportement du controller de *Login*, vous pouvez modifier le code pour rediriger automatiquement l'utilisateur sur la page d'administration lorsque celui-ci appelle "*admin/login*" alors qu'il est déjà connecté.

Si cette étape fonctionne lorsque vous vous connectez, vous êtes automatiquement redirigé sur la page d'administration.

Enfin, lorsque l'utilisateur souhaite se déconnecter, pensez à ajouter une méthode type *logout* qui :

1. détruit la session

2. redirige vers la page d'accueil du portfolio

Voici le pseudo code de la méthode logout :

```
/**
* Disconnects an user from administration
*/
public static Result logout() {
 // some instructions...
}
```

Attention, n'oubliez pas :

1. d'ajouter un lien Logout dans le menu
2. de configurer la route dans le fichier route
3. décommenter le lien dans la vue main.scala.html

Vous devez être en mesure de vous déconnecter de l'application

Liens à consulter :

Pour créer un Controller et lui ajouter des actions, voir ce lien

Ebean : ce lien

Pour rediriger l'utilisateur, consulter le bas de la page à l'url suivante

Exercice 3 : Ajout d'une image

But : Une fois connecté en tant qu'administrateur, vous devez avoir la possibilité d'ajouter des images.

Nouvelles notions abordées :
- **Création d'un modèle**
- **Intéragir avec une base de données**

Une fois connecté, vous avez accès au panneau d'administration des images à partir du lien "Partie Administrateur". Cette vue s'appelle *listImage.scala.html*. Vous la trouverez déjà présente dans le dossier *app/views*.

Afin d'ajouter une image, vous devez être capable de la représenter en base de données. Pour cela, il faut ajouter un modèle *Image* (*app/models/Image.java*) pour représenter une image en base. Pour ce T.P. nous considérons qu'une image est composée d'un :

1. *id* qui est l'identifiant de l'image qui devra être auto-généré
2. *path* étant le chemin d'accès vers l'image à partir du dossier public
3. *listComment*, la liste des commentaires associés à l'image

N'oubliez pas d'écrire les *getters* et *setters*.

Si vous avez correctement créé votre modèle Image, au prochain lancement de l'application, Play vous demandera de mettre à jour votre base de données.

Vous allez maintenant remplir la méthode *addImage* du Controller *ImageAdmin*. Celle-ci est appelée lorsque le formulaire contenu dans la vue *listImage.scala.html* est soumis. Cette méthode vous permettra ainsi d'ajouter des images à votre base de données.

Tout d'abord, vous devrez décommenter la partie servant à stocker l'image dans le dossier d'upload. Puis, il faudra ajouter en base de données la référence de cette image à l'aide de Ebean, c'est-à-dire le chemin qui lui correspond dans le dossier upload depuis le projet.

Maintenant, pour voir vos images ajoutées, créez une nouvelle action nommée listImages() dans le controller *ImageAdmin* qui retournera une liste des images stockées en base à la vue index.scala.html.

Dernière étape, décommentez la partie centrale de cette vue et toutes les images que vous uploaderez apparaitront sur la page d'accueil.

```
<!-- commentaire à supprimer dans index.html.scala -->
@***
<!-- code Twirl -->
***@
```

Si tout se déroule correctement, vous pouvez maintenant ajouter une Image. Celle-ci doit alors apparaître sur la même page.

Lien à consulter :

Vous pouvez vous aider de l'url suivante afin de mieux appréhender les Modèles

Exercice 4 : Supprimer une image

But : Vous devez écrire du code afin de supprimer une image à partir de l'interface administrateur

Nouvelles notions abordées :
- **Création d'une route avec paramètres**

Il faut commencer par créer la méthode dans le controller *ImageAdmin* . Une fois cette étape réalisée, vous devez associer cette action à une route. Attention, vous devez supprimer la bonne image. Il faut donc penser à récupérer l'identifiant de celle-ci. Vérifiez également que l'image soit supprimée du dossier *public/upload/*.

Afin que la suppression soit fonctionnelle, il faut décommenter dans la vue *listImage.scala.html* la partie centrale :

```
<!-- commentaire à supprimer dans listImage.html.scala -->
@***
@helper.form(routes.ImageAdmin.removeImage()) {
      <!-- code Twirl →
}
***@
```

Exercice 5 : Supprimer un commentaire

But : A partir de l'interface administrateur, vous devez être capable de supprimer des commentaires

Nouvelles notions abordées :
- **Passage de paramètre dans l'url**

Dans un premier temps, il faut créer la méthode qui permettra de supprimer un commentaire associé à une image. Vous pourrez trouver sa signature dans le Controller *CommentAdmin*. Vous devrez modifier le corps de la méthode.

Une fois cette étape réalisée, vous devez associer cette action à une route. Attention, vous devez supprimer le bon commentaire. Il faut donc penser à récupérer l'identifiant de celui-ci.

Afin que la suppression soit fonctionnelle, il faut ajouter dans la vue *listImage.scala.html*, la ligne suivante :

```
<ul class="commentList list-group">
@img.getListComment.map { comment =>
      <li role="presentation">
          <span>@comment.getText
               <a href="@routes.CommentAdmin.delete(comment.getId)">X</a>
          </span>
      </li>
}
</ul>
```

Vous devez maintenant être capable de supprimer un commentaire à l'aide de la croix.

Liens à consulter :

Pour le passage de paramètre, vous pouvez vous aider de la session "The routes file syntax" de l'url suivante