**L’internationalisation dans Mbinkus Farm**

**Le problème**

Quand on développe une application, on ne sait généralement pas tous les utilisateurs qui l’utiliseront. De ce fait, on ne connait pas leur langue, est-ce le Français ou l’Anglais ou … ?

Alors, comment faire ?

On peut développer l’application pour chaque langue. Dans ce cas, il faudra refaire toute l’application dans chaque langue et cela demande beaucoup de temps et d’énergie. Imaginons que notre application doit être utiliser dans plus de dix pays qui ont chacun leur langue. On va très vite en besogne.

Avons-nous une autre solution ?

Eh bien oui !

Si notre application s’adaptait à son environnement d’exécution (la langue, la monnaie, …) ? ce serait préférable que d’avoir plusieurs applications. Imaginons qu’un bug soitt détecté sur l’une d’entre elles, on devrait le résoudre dans toutes les autres (Elles font toutes les mêmes choses, c’est juste la langue qui change).

C’est ce qu’on appelle **l’Internationalisation.**

Notre pays étant bilingue (Français et Anglais), nous avons voulu l’implémenter sur notre application « Script Farm ». A ce jour l’implémentation n’est effective que sur la page de Login, c’est de celle-ci que sera inspirée ce document. Nous utiliserons des ResourceBundle et des fichiers .properties. Jamais entendu parler ?

(Pour ceux qui ont répondu oui, Il faut cravailler ! En tout cas dans la suite nous allons codifier …)

A vos IDEs, on commence.

**L’internationalisation en Java**

Avant toute chose, pourquoi en Java ?

Notons d’abord que L’internationalisation est possible dans la majorité des langages de programmation.

Maintenant que c’est fait, pour répondre à la question, nous développons « Script Farm » en Java. J’espères que tout est clair maintenant ? 😊

Pas vraiment, comment on s’y prend ?

Tout doux, on y arrive. Mais avant, parlons de l’objet « Locale ».

Qu’est ce que c’est ?

Un Objet de type « Locale » identifie une langue et un Pays.

Comment le construit-on ?

Son constructeur attend deux paramètres :

-Le code langue (deux caractères minuscules conformes à la norme ISO-639 : exemple "de" pour  
l'allemand, "en" pour l'anglais, "fr" pour le français, ...).

-Le second est le code pays (deux caractères majuscules conformes à la norme ISO-3166 : exemple : "DE"  
pour l'Allemagne, "FR" pour la France,"US" pour les Etats Unis, etc ...). Ce paramètre est obligatoire : si le pays n'a pas besoin d'être précisé, il faut fournir une chaîne vide.

**Locale locale\_US = new Locale(“en”,”US”);**

**Locale locale\_en = new Locale(“en”,””);**

On peut aussi fournir un 3e paramètre pour préciser d'avantage la localisation par exemple la plate-forme d'exécution (il ne respecte aucun standard car il ne sera défini que dans l'application qui l'utilise) :

**Locale locale\_windows = new Locale(“fr”,”FR”,”WINDOWS”);**

Il y a un problème !!! Devons-nous mémoriser tout ça ?

Malheureusement, la classe Locale définit des constantes pour certaines langues et pays.

En ce qui concerne les pays, on peut avoir :

**Locale locale\_japon = Locale.JAPAN ;**

Ce qui est équivalent à :

**Locale locale\_japon = new Locale(“ja”,”JP”);**

On peut aussi vouloir spécifier les langues seulement

**Locale locale\_japon = Locale.JAPANESE ;**

Ce qui est équivalent à :

**Locale locale\_japon = new Locale(“ja”,””);**

Un cadeau pour la route 😊

Comment connaitre le Locale par défaut :

**Locale de = Locale.getDefault();**

Comment connaitre la liste des Locales disponibles :

**Locale liste[] = DateFormat.getAvailableLocales();  
for (int i = 0; i < liste.length; i++)  
{  
System.out.println(liste[i].toString());  
// toString retourne le code langue et le code pays séparé d'un souligné  
}**

Merci. Mais, jusqu’ici je ne vois toujours pas comment l’internationalisation se fait !

Si vous vous rappelez, on a dit qu’on utilisera des ResourceBundle et des fichiers .properties

On va arrêter de vous perdre du temps 😊 Entrons maintenant dans le vif du sujet.

**Les fichiers .properties**

Pour que notre application s’adapte aux langues, il faut traduire tous les mots et phrases qu’on affiche. Je suis désolé, mais si vous voulez d’une telle application, c’est le prix à payer. Sinon, vous pouvez la redévelopper pour chaque langue. Et même là, vous aurez toujours besoin de traduire chaque mot et phrase qui sera affiché.

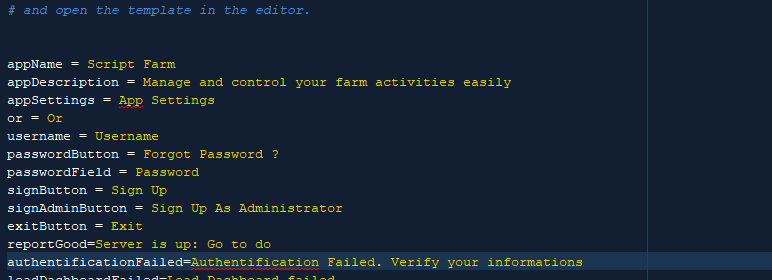
En traduisant, nous allons créer des fichiers .properties : Un pour chaque langue et un qui sera choisi par défaut.

Les lignes d’un fichier .properties sont soient :

-Un commentaire. Dans ce cas, la ligne commence par un « # ».

-Une paire clés/valeur séparées par un « = »Ex : appName = Script Farm

-vides.



*Exemple du contenu d’un fichier .properties*

Il doit avoir au moins autant de fichier .properties que de langue : Chaque langue a son .properties et il y a le .properties par défaut. Il y a une façon particulière de nommer ces fichiers. Nous y reviendrons.

**Les ResourceBundle**

Une ResourceBundle est une classe java rattachée à un fichier .properties. En fait plusieurs :

* Le fichier qui sera utilisé par défaut. Ex : login.properties
* Ceux qui seront utilisés pour d’autres langue. Ex : login\_en\_EN.properties, login\_fr\_FR.properties , login\_fr.properties.

**NB : Ces fichiers doivent être dans le même répertoire.**

Comme vous pouvez le constater, les autres fichiers ont des noms particuliers.

Il y a-t-il une logique ?

Forcement !

Pour ceux qui ne l’on pas remarquée, la voici :

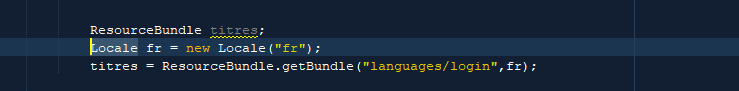
* Il commence exactement comme le nom du fichier par défaut.
* Ensuite on ajoute le code la langue et/ou du pays séparés par un espace.

Ex : Le fichier qui sera utilisé pour traduire l’application en Allemand sera : login\_de.properties.

Un ResourceBundle nous permet selon la langue courante de sélectionner le fichier qui sera utilisé pour la traduction.

Il suffit de faire

**ResourceBuilder.getBundle(" « chemin\_d’acces »/ « Nom\_fichier\_par\_defaut »", « Locale ») ;**

« Locale » désigne l’instance de la classe Locale correspondante à la langue qu’on veut utiliser.

*Initialisation d’un ResourceBundle*

Naturellement, il faudra importer toutes les packages nécessaires. Comme chacun de nous utilise un IDE, nous ne nous attarderons pas dessus ; l’IDE fera son job en vous disant quels packages importer.

Sachez que java.util contient tout le nécessaire.

**L’utilisation d’un ResourceBundle**

A présent que nous avons initialiser notre ResourceBundle, comment l’utiliser ?

On avait dit que les .properties contient des couples clés/valeur.

Oui et alors ?

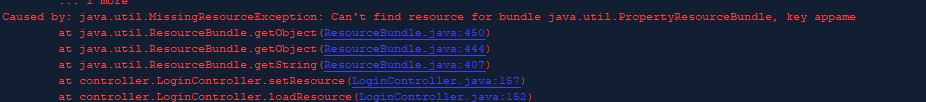
Les clés devront être identiques dans toutes les fichiers .properties rattachés au fichier .properties par défaut.

Pour implémenter l’internalisation, il suffira pour chaque texte à afficher de l’associer à une clés et de donner la signification de ce texte dans les différentes langues(dans chaque fichier .properties). Au lieu d’écrire le texte, on utilisera la méthode getString() et on lui donnera la clés en paramètre.

**titres.getString(«appName ») ;**

Si la clé fournie en paramètre n’est pas disponible, une MissingResourceException est levée.

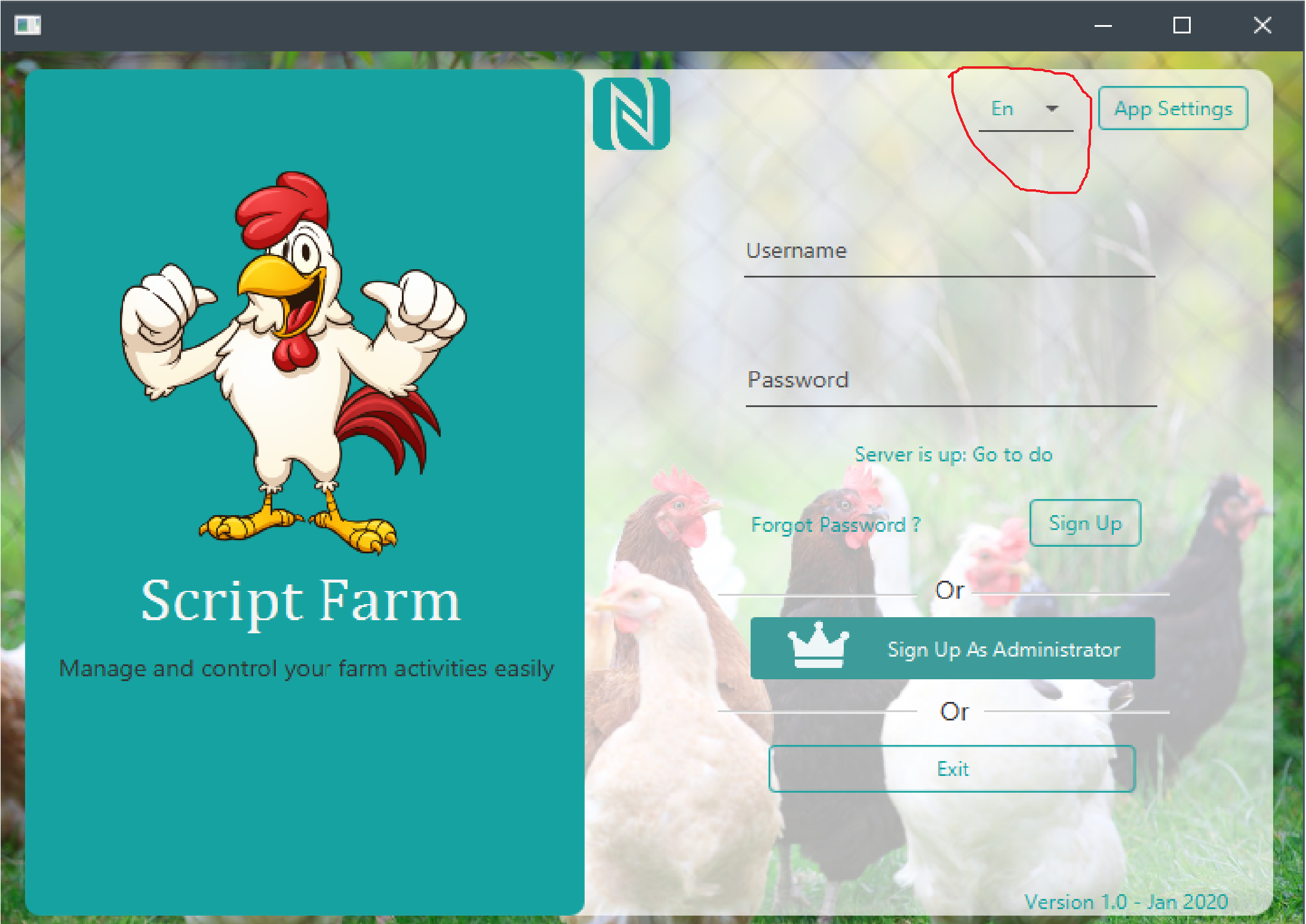
Ces fameuses lignes rouges que nous connaissons crès bien. Cette image vous parle bien sûr !



*MissingResourceException*

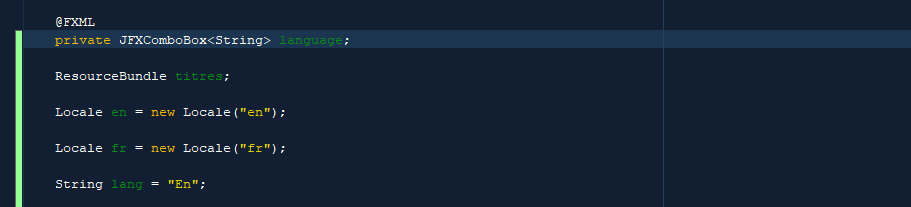
**L’internationalisation et Script Farm**

Dans la vue du login, on a ajouté un comboBox qui nous permet de sélectionner la langue :



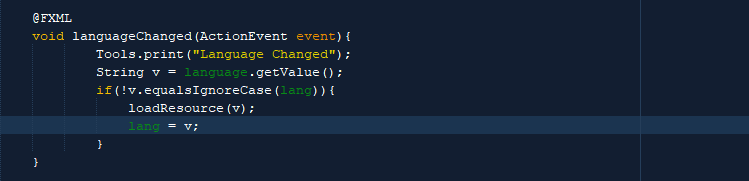
*login*

Dans le controller, on initialise notre ResourceBundle et nos variables Locales qui vont identifier nos langues (Français et Anglais) et une variable String qui nous permettra de connaitre la langue utilisée à tout instant.

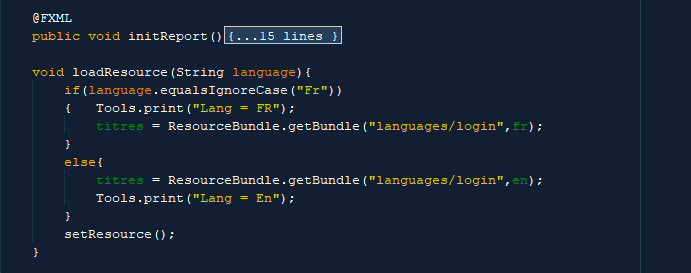


Pour gérer l’internationalisation, on a créé trois méthodes :

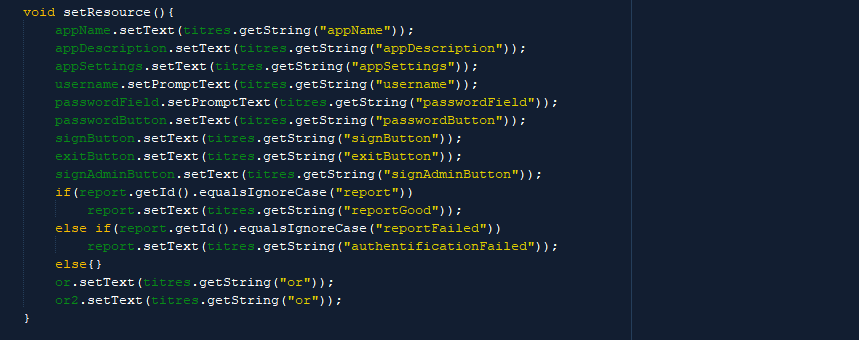
* Un listerner sur la comboBox



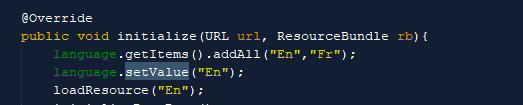
* Une méthode qui charge la ressource correspondante en fonction de la langue qui lui est envoyée



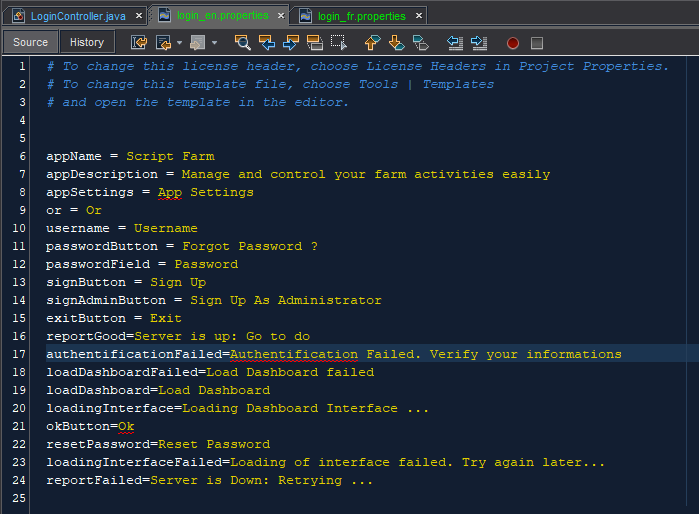
* Une méthode qui applique la ressource sur toute l’interface

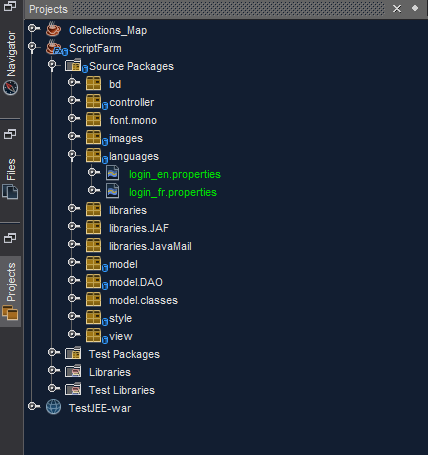
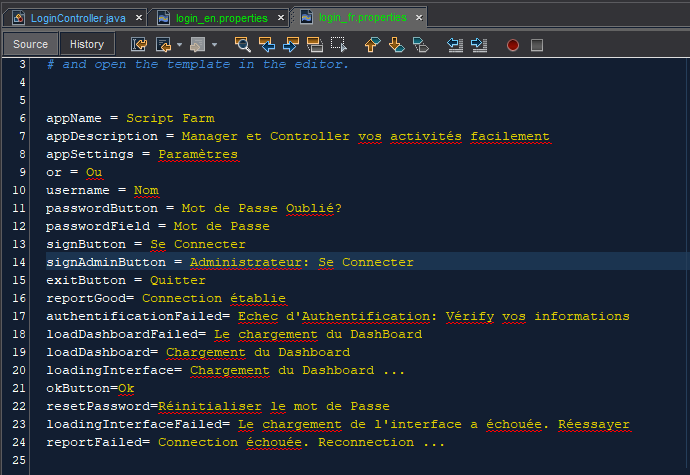


Dans la méthode initialize du controller, on initialise le comboBox , et on charge la resource



Mais, où sont les fichiers .properties ?

Ils sont là Bro. Juste en dessous.



**NDEMA @The Script Farm Team**