Rails, une alternative à J2EE

http://www.rubyonrails.org/

Présentation

Ruby on Rails, ou RoR ou Rails, est un framework web basé sur le design pattern MVC et utilisant le langage Ruby.

Ruby est un langage de programmation interprété orienté objet. Toute donnée est un objet, y compris les types primitifs.

Rails a été conçu avec l'idée de respecter les deux principes « Ne pas se répéter » et « Convention plutôt que Configuration ».

Le premier prône la réutilisation du code déjà existant.

Le deuxième force les développeurs à respecter des conventions de nommage. C'est à première vue une contrainte, mais cela va permettre à Rails de prendre en charge la configuration de l'application au lieu de laisser cette tâche aux développeurs.

Le framework MVC Rails est complet et propose des outils pour chacune des couches de l'application. le modèle est géré par le composant ActiveRecord, le contrôleur et la vue par Action Pack, les web services par le composant Action Web Service. Ajax est intégré avec le composant Prototype.

Installation

Télécharger sur le site RubyInstaller l'installation tout-en-un de Ruby.

Installer Ruby en suivant les recommandations de l'installeur.

Pour tester que Ruby est bien installé, dans une invite de commandes, taper la commande irb; irb est un interpreteur de commandes ruby, comme on a pour le langage Caml ou des langages fonctionnels comme Scheme ou Lisp.

Passons à l'installation de Rails. Rails est un framework MVC pour Ruby. La procédure d'installation est la même que pour les autres packages de Ruby. Dans une invite de commandes, tapez la commande:

```
gem install rails -remote
```

gem téléchargera et installera lui même le package rails ainsi que tous les packages dont dépend Rails.

Tout est prêt pour mettre en place notre prémiere application Rails.

Configuration de l'application

L'application est un annuaire gérant la liste des collaborateurs de l'entreprise.

Pour le stockage des données, nous créeons une base de données Mysql annuaire.

Dans cette base de données, nous créeons une table collaborateurs. (le pluriel est important pour le nom de la base, Rails imposant certaines régles de nommage)

	Field Name	Datatype	Len	Default	PK?	Binary?	Not Null?	Unsigned?	Auto Incr?	Zerofill?
*	id	int	11		☑	☐	☑	☐	☑	☐
	nom	varchar	255		☐	☐	☑	☐	☐	☐
	prenom	varchar	255		☐	☐	☑	☐	☐	☐
	dateNaissance	date			☐	☐	☐	☐	☐	☐
	telephone	varchar	255		☐	☐	☐	☐	☐	☐
	mail	varchar	255		☐	☐	☐	☐	☐	☐
					☐	☐	☐	☐	☐	☐

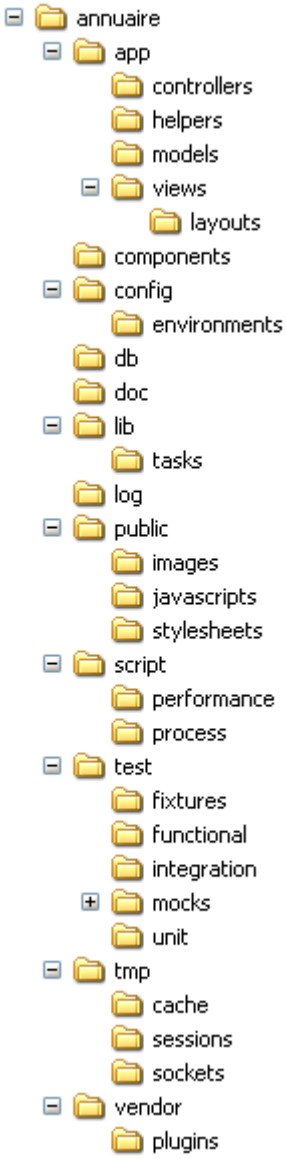
Rails impose d'avoir un champ id comme primary key auto-incrémentée.

Ensuite, dans une console, placez-vous dans le répertoire de votre choix où l'application sera placée et tapez la commande :

```
rails annuaire
```

Rails va créer pour nous le squelette de notre application.

Allez dans le répertoire *annuaire* nouvellement créé.



Dans le répertoire app se trouveront nos fichiers ruby. Rails va nous aider pour les créer, notamment il va génerer pour nous les fichiers de mapping avec la base.

Dans le fichier config/database.yml se trouvent les informations pour se connecter à la base. Par défaut, rails configure l'accès à 3 bases, une pour le développement, une pour la production et une pour les tests. Seule celle pour le développement nous intéresse pour le moment.

Au besoin, remaniez les informations comme username, password, host et database. Dans notre cas, database est modifié. Le nom de la base est annuaire et non annuaire_development.

Pour créer ensuite un CRUD permettant de manipuler les données de notre base avec les opérations de base (Create, Read, Update, Delete), dans une console, allez dans le répertoire annuaire et tapez la commande :

```
ruby script\generate scaffold Collaborateur
```

le script ruby generate va generer pour nous le modele, le controlleur et les vues de notre CRUD.

Il ne reste alors plus qu'à le tester.

Lançons pour ça le serveur d'applications Rails; tapez dans une console :

```
ruby script\server
```

Dans votre navigateur web, allez à l'URL : http://localhost:3000/collaborateurs

Vous avez déjà un CRUD basique qui marche sur le serveur d'applications WebRick, idéal pour le

développement. Lorsque vous souhaiterez passer en production, il sera possible de passer sur un serveur Apache.

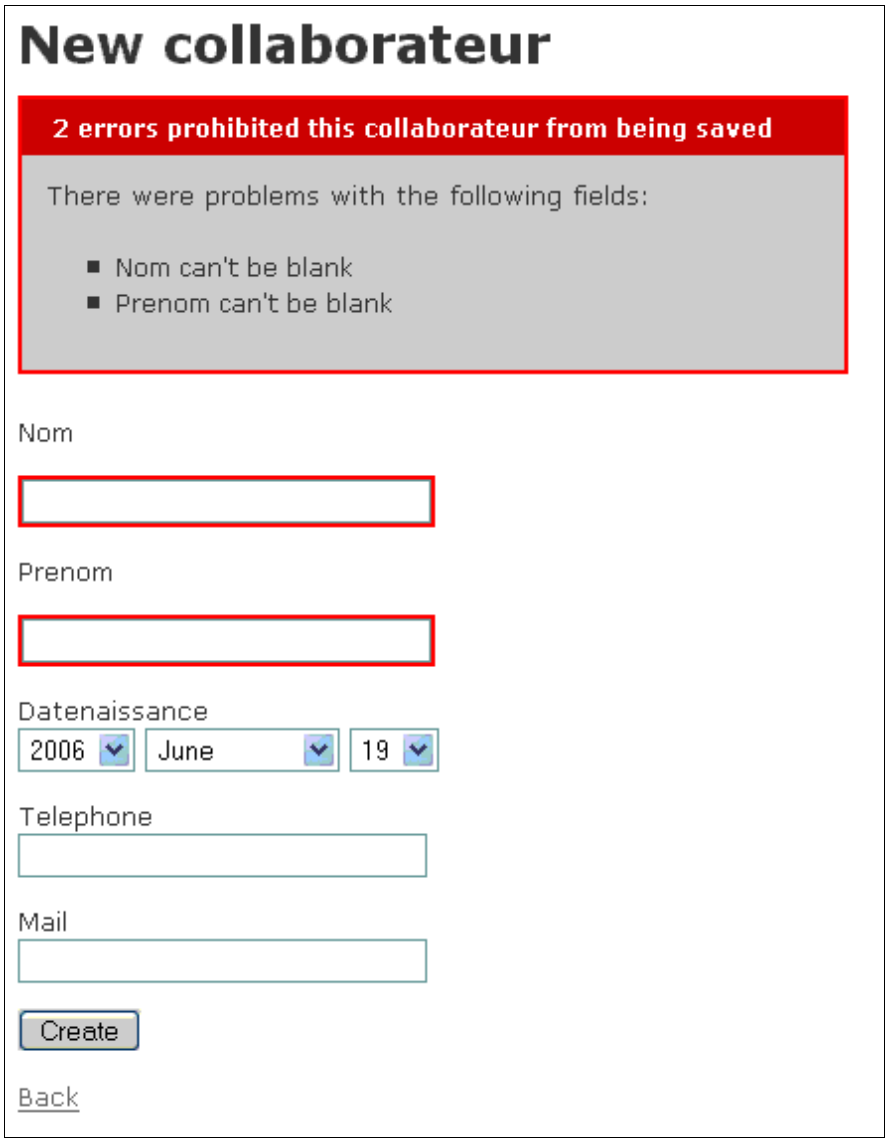
Validation de formulaire

Pour rester cohérent avec la base de données, nous devons exiger que les champs du formulaire Nom et Prénom de la page edit ou new soient non nuls.

Modifions le fichier app/models/controlleur.rb:

```
class Collaborateur < ActiveRecord::Base
      validates_presence_of :nom, :prenom
end
```

Maintenant, si le formulaire est validé avec un des champs nom ou prenom vide, nou aurons le message d'erreur suivant :

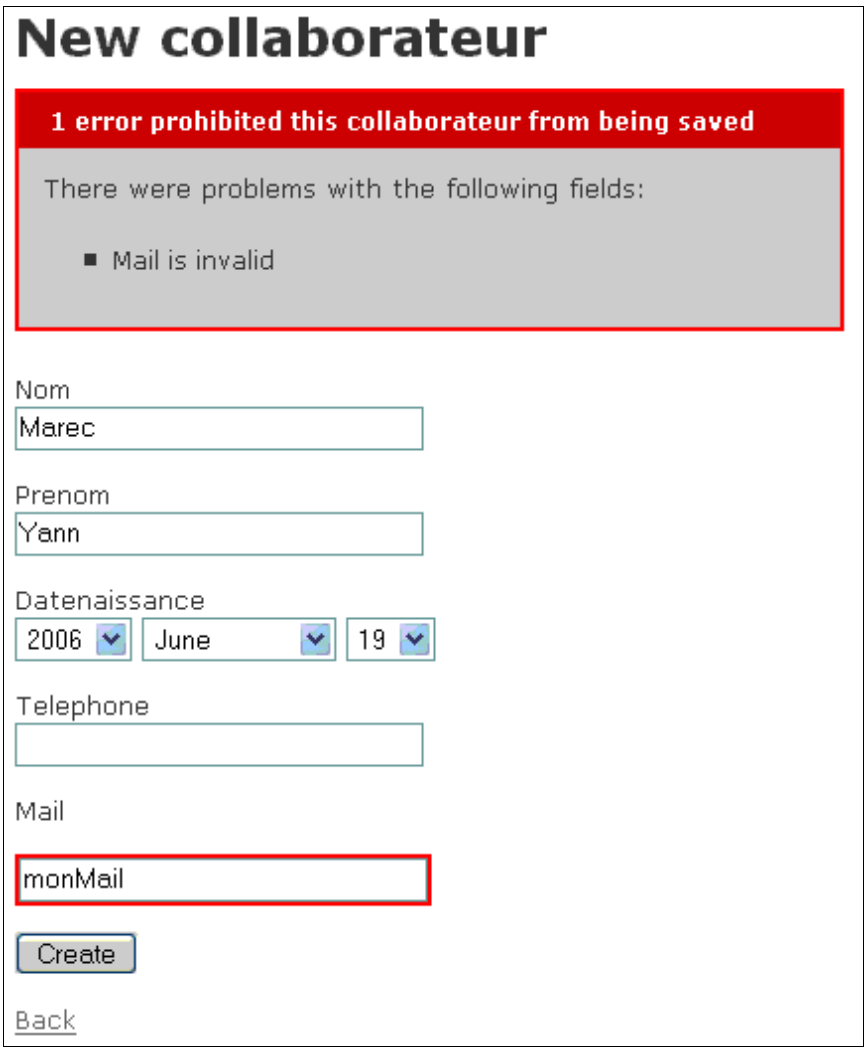


Nous souhaiterions un contrôle sur le champ mail pour qu'il respecte le format des adresses mails.

Modifions le fichier app/models/controlleur.rb:

```
class Collaborateur < ActiveRecord::Base
      validates_presence_of :nom, :prenom
      validates_format_of :mail, :with => /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})$/
end
```

on aura alors ce message d'erreur si jamais le format du mail est invalide :



Modification des vues

On peut vouloir modifier l'affichage d'une des pages, par exemple celle listant les collaborateurs.

Rails nous a généré les fichiers .rhml qui sont des pages html agrémentées de code ruby.

Modifions donc le fichier app/views/collaborateur/list.rhtml pour que la liste n'affiche que le nom et le prenom du collaborateur et que ce nom+prenom soit un lien vers la fiche complete du collaborateur:

```
<h1>Listing collaborateurs</h1>

<table>
<% for collaborateur in @collaborateurs %>
  <tr>
      <td><%= link_to collaborateur.nom + ' ' + collaborateur.prenom , :action =>
'show', :id => collaborateur %></td>
      <td><%= link_to 'Edit', :action => 'edit', :id => collaborateur %></td>
      <td><%= link_to 'Destroy', { :action => 'destroy', :id => collaborateur },
:confirm => 'Are you sure?', :post => true %></td>
  </tr>
<% end %>
</table>

<%= link_to 'Previous page', { :page => @collaborateur_pages.current.previous } if
@collaborateur_pages.current.previous %>
<%= link_to 'Next page', { :page => @collaborateur_pages.current.next } if
@collaborateur_pages.current.next %>

<br />
```

```
<%= link_to 'New collaborateur', :action => 'new' %>
```

Pour que le champ date de notre formulaire soit plus « francophone », il faut modifier le fichier app/views/collaborateur/_form.rhtml et mettre + d'options sur le date_select :

```
<%= date_select
      'collaborateur',
      'dateNaissance',
      :order => [:day, :month, :year],
      :use_month_numbers => true,
      :start_year => 1900,
      :end_year => Date.today.year,
      :include_blank => true%>
```

le date_select va alors chercher l'information dateNaissance dans la table collaborateur. L'ordre des select sera (jour, mois, année), les noms des mois seront remplacés par leur numero, la fourchette des années sera [1900;année en cours] et l'utilisateur a la possibilité de ne pas renseigner l'un des select.

Pour que le format d'affichage des dates soit par defaut jj/mm/yyyy et que les noms des jours et mois soient en français, il faut ajouter à la fin du fichier config/environment.rb cette ligne :

```
require 'overrides'
```

et ajouter le fichier lib/overrides.rb :

```
Date::MONTHS = { 'Janvier' => 1, 'Fevrier' => 2, 'Mars' => 3, 'Avril' => 4, 'Mai' => 5, 'Juin' => 6, 'Juillet' => 7, 'Aout' => 8, 'Septembre'=> 9, 'Octobre' =>10, 'Novembre' =>11, 'Decembre' =>12 }
Date::DAYS = { 'Lundi' => 0, 'Mardi' => 1, 'Mercredi' => 2, 'Jeudi'=> 3, 'Vendredi' => 4, 'Samedi' => 5, 'Dimanche' => 6 }
Date::ABBR_MONTHS = { 'jan' => 1, 'fev' => 2, 'mar' => 3, 'avr' => 4, 'mai' => 5, 'juin' => 6, 'juil' => 7, 'aou' => 8, 'sep' => 9, 'oct' =>10, 'nov' =>11, 'dec' =>12 }
Date::ABBR_DAYS = { 'lun' => 0, 'mar' => 1, 'mer' => 2, 'jeu' => 3, 'ven' => 4, 'sam' => 5, 'dim' => 6 }
Date::MONTHNAMES = [nil] + %w(Janvier Fevrier Mars Avril Mai Juin Juillet Aout Septembre Octobre Novembre Decembre )
Date::DAYNAMES = %w(Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche )
Date::ABBR_MONTHNAMES = [nil] + %w(jan fev mar avr mai juin juil aou sep oct nov dec)
Date::ABBR_DAYNAMES = %w(lun mar mer jeu ven sam dim)

class Time
      alias :strftime_nolocale :strftime

      def strftime(format)
            format = format.dup
            format.gsub!(/%a/, Date::ABBR_DAYNAMES[self.wday])
            format.gsub!(/%A/, Date::DAYNAMES[self.wday])
            format.gsub!(/%b/, Date::ABBR_MONTHNAMES[self.mon])
            format.gsub!(/%B/, Date::MONTHNAMES[self.mon])
            self.strftime_nolocale(format)
      end
end
```

Mapping des relations 1-n

Nous allons ajouter une table fonctions à notre base de données pour illustrer les relations 1-n.

Un collaborateur exerce une fonction et une fonction peut être exercée par plusieurs collaborateurs.

	Field Name	Datatype	Len	Default	PK?	Binary?	Not Null?	Unsigned?	Auto Incr?	Zerofill?	Comment
*	id	int	11		☑	☐	☑	☐	☑	☐	
	titre	varchar	255		☐	☐	☑	☐	☐	☐	
					☐	☐	☐	☐	☐	☐	

Comme pour collaborateurs, nous devons avoir une colonne id comme clé primaire et auto-incrémentée.
De plus, il faut rajouter une colonne fonction_id à la table collaborateurs en tant que int(6).

Dans une console, nous lançons la commande :

```
ruby script\generate scaffold fonction
```

Un CRUD est alors disponible pour les fonctions à l'URL : http://localhost:3000/fonctions
vous pouvez l'utiliser pour y entrer des fonctions qui seront ensuite affectées aux collaborateurs.
Pour faire la jointure entre les deux classes, il faut ajouter une ligne dans chacune d'entre elles :
Dans le fichier app/models/collaborateur.rb :

```
class Collaborateur < ActiveRecord::Base
      belongs_to :fonction

      validates_presence_of :nom, :prenom
      validates_format_of :mail, :with => /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})$/
end
```

et dans le fichier app/models/fonction.rb :

```
class Fonction < ActiveRecord::Base
      has_many :collaborateurs
end
```

Il faut maintenant modifier le contrôleur de collaborateur pour qu'il prenne en compte les fonctions.
On modifie donc le fichier app/controllers/collaborateurs_controller.rb et en particulier sa méthode edit :

```
def edit
    @collaborateur = Collaborateur.find(params[:id])
    @fonctions = Fonction.find_all
end
```

La vue utilisera cet attribut @fonctions pour remplir le select du formulaire.
Logiquement, il faut donc faire de même dans la méthode new :

```
def new
    @collaborateur = Collaborateur.new
    @fonctions = Fonction.find_all
end
```

Il faut finalement ajouter un select au fichier app/views/collaborateurs/_form.rhtml :

```
<p>
      <b>Fonction:</b><br>
      <select name="collaborateur[fonction_id]">
      <% @fonctions.each do |fonction| %>
             <option value="<%= fonction.id %>"
            <%= ' selected' if fonction.id == @collaborateur.fonction_id %>>
            <%= fonction.titre %>
             </option>
      <% end %>
      </select>
</p>
```

et afficher cette nouvelle information dans la fiche d'un collaborateur dans le fichier app/views/collaborateurs/show.rhtml :

```
<p><b>Fonction: </b><%= @collaborateur.fonction.titre %></p>
```

Conclusion

Ce qui saute aux yeux est le temps gagné en utilisant Rails.

Cette application Annuaire a été développé dans un premier temps avec les technologies J2EE.

Je connaissais bien Java, j'avais une premiere experience dans le developpement d'applications J2EE avec servlets et jsp mais ne connaissais ni Hibernate, ni Struts, ni Spring. J'ai mis au moins deux semaines pour apprendre à me servir de ces frameworks et les mettre en pratique pour réaliser l'application Annuaire et écrire le document expliquant mon travail.

Dans un second temps, pour tester Rails, l'application a été redeveloppé entièrement avec Rails.

Je ne connaissais pas Ruby, je ne connaissais pas le fonctionnement de Rails. J'ai réalisé le même travail que j'avais fait avec J2EE, mais cette fois ci en 2 jours.

Le gain de temps est donc clairement évident.

Ce comparatif est fait sur le développement d'une petite application, mais on a des retours de développeurs travaillant sur de grosses applications (comme James Duncan, auteur de Tomcat, Ant et de l'API servlet) affirmant que Rails est un framework très bien pensé qui peut être utilisé sur de gros projets, et facilite le développement et aboutit à un code bien formé et facilement maintenable.