Tutoriel Rails

L3 Informatique—Sécurité Informatique

Université Paul Sabatier Toulouse 3

1 Déploiement

Ajout de librairies :

1. éditer le fichier Gemfile pour ajouter la librairie (la ligne à ajouter est de la forme :

```
gem 'librairie'
```

2. sauvegarder, et télécharger la librairie :

```
bundle install
```

Lancement du serveur Web :

```
rails s
```

Lancement d'une console :

```
rails c
```

2 Interaction avec la base de données

La base de données actuelle est décrite dans le fichier `db/schema.rb`. Ce fichier ne doit pas être édité manuellement. Lorsqu'on veut changer cette base, il faut créer une nouvelle *migration* qui décrit les changements à faire, et ensuite appliquer cette migration. On peut revenir en arrière avec :

```
rake db:rollback
```

Les principales commandes sont :

Création :

```
rake db:migrate
```

Changement dans les tables existantes :

1. Un changement s'appelle une *migration*. On crée une nouvelle migration vide avec :

   ```
   rails g migration nom_migration
   ```

   Dans certains cas (ajout d'une colonne), la migration peut être générée depuis la ligne de commande. Par exemple, pour ajouter une colonne `age` de type `int` à la table `users`, on peut utiliser :

   ```
   rails g migration add_age_to_users age:int
   ```

2. Mettre à jour la base de données :

   ```
   rake db:migrate
   ```

Les migrations permettent aussi de créer de nouvelles tables, mais en général ces tables sont liées à des **modèles**, et seront créées automatiquement lors de la création du modèle. Il faudra cependant toujours faire la mise à jour avec :

```
rake db:migrate
```

3 Modèles et scaffold

3.1 Modèles

Un modèle est défini par une classe. Par convention, le modèle `User` est défini par le fichier `user.rb` dans le répertoire `app/models`.

Création d'un nouveau modèle : La commande :

```
rails g model shoe  user:references size:int:index couleur:string
```

crée un nouveau modèle `Shoe`. Dans la base de données, chaque enregistrement contiendra une clef étrangère (`references`) venant de la table `users`, une valeur `size` de type `int` avec un index sur cette colonne (pour pouvoir récupérer rapidement les chaussures d'une taille donnée), et enfin une chaîne de caractères qui en décrit la couleur. Plus précisément, cette commande a créé la migration qui va créer cette table, et il faut maintenant appliquer cette migration :

```
rake db:migrate
```

Utilisation des relations de la BD dans l'application : — si l'enregistrement contient une référence, on utilise une relation `belongs_to` pour accéder à l'enregistrement référencé :

```
class Shoe < ActiveRecord::Base
  belongs_to :user
end
```

— Si un utilisateur peut avoir une seule chaussure, on écrira dans `app/models/user.rb` :

```
class User < ActiveRecord::Base
  has_one :shoe
end
```

et si un utilisateur peut en avoir plusieurs, on écrira plutôt :

```
class User < ActiveRecord::Base
  has_many :shoes
end
```

(notez le passage du singulier au pluriel).

Création d'un enregistrement : 1. ouvrir la console :

```
rails c
```

2. créer la nouvelle instance :

```
chaussure=Shoe.new({user: User.first, size: 36, couleur: 'rouge'})
```

(User.first est le premier enregistrement trouvé dans la table `users`)

3. sauvegarder cette instance :

```
chaussure.save
```

on aussi peut le faire en une seule étape :

```
Shoe.create({user: User.first, size: 36, couleur: 'rouge'})
```

Retrouver des modèles : On fait une recherche sur les modèles en utilisant `where`. Par exemple (console rails) :

```
taille36=Shoes.where({size: 36})
```

(`where` rend un tableau possiblement vide).

3.2 Scaffold

Dans l'énoncé du TP, les tables correspondant uniquement à des modèles sont en rouge. Celles qui sont en bleues peuvent être manipulées (ajout/suppression d'enregistrement, visualisation, édition) à travers l'interface Web.

Contrôleur : un contrôleur contient des méthodes à exécuter. Par défaut, ce sont les méthodes ReST : show, new (formulaire de création), create (création), edit (formulaire de modification), update (modification), index, et destroy. Par convention, pour le modèle `Shoe`, le contrôleur est `app/controllers/shoes_controller.rb`.

Routage : le fichier `config/routes.rb` contient la liste des actions à faire lors de la réception d'une requête HTTP. Par défaut, la déclaration :

```
resources :shoes
```

va créer 7 couples (url+verbe http) qui seront traduites en appels de méthode dans le contrôleur (par exemple, le couple $('/shoes', get)$ va conduire à appeler la méthode `index` du contrôleur `shoes_controller.rb`.

Vues : les méthodes du contrôleur assignent des variables globales (dont le nom commence par @) qui sont ensuite utilisées pour construire la réponse. Pour chaque méthode par défaut, la réponse est construite à partir d'un des fichiers du répertoire `app/views/shoes/`. Les fichiers `.html.erb` mêlent de l'html et du ruby de 2 manières :

- lorsque le résultat de l'évaluation du code ruby doit être inséré, on utilise la balise `<%=` ;
- sinon, on utilise la balise `<%` ;

Tous ces fichiers sont générés automatiquement si, au lieu d'utiliser :

```
rails g model shoe user:references size:int:index couleur:string
```

on utilise :

```
rails g scaffold shoe user:references size:int:index couleur:string
```

Notez que dans ce cas, comme on modifie la configuration (le fichier `config/routes.rb`), il faut relancer le serveur.