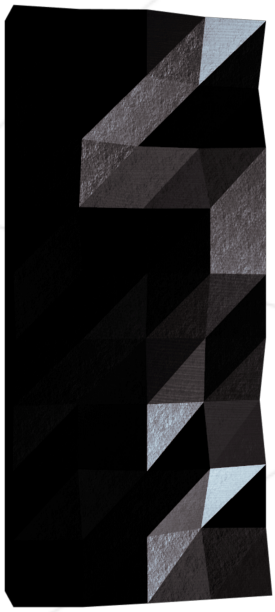

Formation Ruby on Rails - 3

Avancées

Résumé: Une journée sur les pratiques appliquées du dev web. Vous êtes mis en situation via des 'stories' qui mèneront à la création d'un site de e-commerce basique.

Version: 1

Table des matières

Chapitre I

Préambule

Écrire une histoire utilisateur (scenario, story)

`Une histoire utilisateur est une suite d'actions effectuées par un utilisateur de notre application suivies d'une ou plusieurs assertions (test) validant que notre histoire s'est bien déroulée`.

Pour expliquer en langage courant, une histoire ressemble à un problème de mathématiques dans lequel on a :

- des hypothèses
- un élément perturbateur
- quelque chose à démontrer

Exemple de problème mathématique (version collège) :

- Soit 2 personnes (Pierre et Jean)
- Pierre possède 3 bananes
- Jean possède 2 fois plus de bananes que Pierre quand Jean mange une banane
- alors combien de bananes possède Jean ?

Transformons cet exemple de manière mathématique (version prépa, merci M. Bool) :

- Soit 2 personnes (Pierre et Jean)
- Pierre possède 3 bananes
- Jean possède 2 fois plus de bananes que Pierre quand Jean mange une banane
- alors prouvons que Jean possède 5 bananes.

Si maintenant nous voulons écrire une histoire utilisateur validée par notre système, nous écrirons :

- Soit 2 personnes (Pierre et Jean)
- Pierre possède 3 bananes
- Jean possède 2 fois plus de bananes que Pierre quand Jean mange une banane
- alors Jean devrait posséder 5 bananes.

C'est ce qu'on appelle un test : Jean **devrait** posséder 5 bananes.
Si nous écrivons des tests, c'est parce que notre système **doit** se comporter comme tel. Cela nous permet de valider que notre application réagit bien TOUJOURS de la même manière, même après de nombreux mois / années (donc après de nombreuses modifications).

Docs, sources et references :

- Rspec, cucumber book
- Le wiki du github de cucumber
- Le site de cucumber
- Motivational

Chapitre II

Règles communes

- Votre projet doit être réalisé dans une machine virtuelle.
- Votre machine virtuelle doit avoir tout les logiciels necessaire pour réaliser votre projet. Ces logiciels doivent être configurés et installés.
- Vous êtes libre sur le choix du systmème d'exploitation à utiliser pour votre machine virtuelle.
- Vous devez pouvoir utiliser votre machine virtuelle depuis un ordinateur en cluster.
- Vous devez utiliser un dossier partagé entre votre machine virtuelle et votre machine hote.
- Lors de vos évaluations vous allez utiliser ce dossier partager avec votre dépot de rendu.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre III

Règles spécifiques de la journée

- Tout le travail de la journée sera des versions ameliorées de la même application rails.
- Toute addition au Gemfile fourni est interdite.
- Toute variable globale est interdite.
- Vous devez rendre une seed en concordance avec les fonctionalités presentes. Lors de votre soutenance votre correcteur doit pouvoir visualiser votre travail.
- rubycritic doit vous decerner un score au moins de 89/100
- vous devez impérativement gérer les erreurs. Aucune page d'erreur rails ne sera tolérée en correction.

`Tips:` Si vous voulez que vos corrections soient plus simples, rapides et sympas, écrivez des tests en rapport avec les stories ! Ca simplifie la vie à tout le monde et c'est une habitude primordiale, que dis-je, IMPERATIVE à prendre pour votre avenir.

Chapitre IV

Exercice 00 : MySQL is a bad habit

	Exercice : 00
Exercice 00 : MySQL is a bad habit	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Commençons par un peu d'AdminSys. Installez `postgresql` sur la machine virtuelle et créez un template et un user. De ce fait, vous pourrez créer une application rails nomée 'acme' utilisant le gemfile que vous trouverez dans la tarball `d07.tar.gz`

Vous devez faire en sorte que la commande "rake db :create" passe sans erreurs.

`Story:`

- L'application s'appele "acme". Je veux pouvoir mettre en ligne l'application sur un hébergeur en ligne (de votre choix).

Chapitre V

Exercice 01 : You Sir ?

	Exercice : 01
Exercice 01 : You Sir ?	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Vous avez une DB, dans une app toute fraiche. Hier vous avez créé une authentification à la main, mais dans la vraie vie on ne fait pas comme ca.
En effet, meme pour un bloguinet (c'est un petit blog), un serveur est présent, et `doit` être protegé.

Pour ce faire, on utilise la `très` bonne librairie `devise` qui, outre faire le petit dèj (à ce stade, le café est déjà trop loin), gère les authentifications.

Si vous inspectez le Gemfile, vous verrez qu'elle est deja présente. Il vous suffit maintenant de l'installer et de la faire gérer un model "User" de sorte que votre seed puisse éxécuter les commandes suivantes :

```
User.create!(bio: FFaker::HipsterIpsum.paragraph,
name: 'admin',
email:'admin@gmail.com',
password:'password',
password_confirmation: 'password' )
```

`Stories:`

- Un utilisateur peux créer un compte avec un mot de passe (nécessaire), un nom (nécessaire), un email(nécessaire) et une biographie (optionelle).
- Il peut re-editer tout ces champs apres creation du compte via l' application (configurez vos "strong parameters")

Chapitre VI

Exercice 02 : Get me something to sell

	Exercice : 02
Exercice 02 : Get me something to sell	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Créez des produits à vendre et des marques, et comme tout produit outre une description tres avantageuse et vantant des mérites que la physique réfute, il faut une image... et c'est là notre problème.

En effet, pour permettre correctement l'upload de fichier d'image, on va utiliser la gem `carrierwave` (the classier solution), rien de trop compliqué jusque là. Le hic maintenant, c'est qu'un hébérgement gratuit supporte trèèès mal le stock de données qui peuvent être volumineuses.

C'est à cet effet que figure dans le Gemfile, une gem appelée cloudinary. Vous devez creer un compte gratuit chez cloudinary, et vous octroyer par là, un espace de stockage distant spécialisé dans les images et autres medias (un CDN quoi).

À ce stade, votre seed peut executer :

```
50.times do |tm|
   mk = Brand.create!(name: FFaker::Product.brand,
                avatar: open(FFaker::Avatar.image))

   50.times do |tw|
      Product.create!(name: FFaker::Product.product,
                  pict: open(FFaker::Avatar.image),
                  description: FFaker::HipsterIpsum.paragraph,
                  brand_id: mk.id, price:price.sample)
   end
end
```

`Stories:`

- En se connectant sur le site de l'application, on voit un catalogue listant des produits.
- Chaque produit est constitué d'un nom, d'une image, d'une description d'une marque et d'un prix.
- Une marque a un nom et une image.
- Peu importe la taille de l'image uploadée, la page des produit doit afficher les 2500 images, dans une version thumbnail, afin de charger rapidement.
- On peut créer et/ou editer tous les champs en ligne des marques et des produits.
- À terme, des roles seront attribués, déterminant qui peux éditer quoi.

Chapitre VII

Exercice 03 : Panier

	Exercice : 03
Exercice 03 : Panier	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Nous avons besoin de faire un panier, un `Cart` qui va se voir associer des copies des produits sous forme de `CartItem`, copiés en `OrderItem` et rassemblés dans un `Order` lors de la validation du panier.

Ces objets particuliers ne nécéssitant pas de CRUD à proprement parler, seul un model leur est utile. Cependant, des fonctionalités devront être placées dans un "Concern" afin de pouvoir inclure des méthodes relatives au panier dans d'autres controleurs comme celui des "Products". Fabriquez vous une méthode current_cart qui se base sur la session_id.

Pensez aussi à détruire les objets et enregistrements inutiles, par exemple lors de l'annulation d un panier, les 'CartItems' associés doivent être détruits.
`Stories:`

- Dans la page catalogue, un encart 'panier' est présent.
- On peut y ajouter des items en cliquant sur le bouton "Ajouter au panier" present sur chaque article.
- L'utilisateur peux commencer une commande, remplir son panier et fermer le navigateur. Lors de son retour sur le site son panier est rechargé.
- L'utilisateur peux augmenter et diminuer le nombre de chaque item dans son panier.
- Les lignes de panier affichent un type d'item, sa quantité, un bouton plus et un bouton moins, ainsi que le prix sur la formule : quantité * prix.

- L'utilisateur peut annuler un panier et le rendre vide avec un bouton.
- Un bouton "Checkout" affiche un recapitulatif de la commande avec le prix total.

Chapitre VIII

Exercice 04 : One panel to rule them all

	Exercice : 04
Exercice 04 : One panel to rule them all	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Vous devez maintenant créer un paneau d'administration digne de ce nom.
Dans le Gemfile, une gem rails_admin est présente, allez voir la doc et initialisez la.

On peut y accéder (pour l'instant) des qu'on possede un compte. Creez sur la page catalogue un link qui mène au dashboard fraichement disponible.

`Stories:`

- Un utilisateur enregistré peux se connecter et aller sur le panneau d'administration du site.
- De là, on peut éditer et visualiser l'ensemble des données du site.

Chapitre IX

Exercice 05 : One account to rule them all

	Exercice : 05
Exercice 05 : One account to rule them all	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Il est, vous me l'accorderez, plutot inutile de disposer d'un panneau d'administration, si tout le monde a la possiblité d'y faire sa loi, pour peu qu'il ait pris la peine de s'y inscrire.

Pour cette occasion, j'ai inclus la gem cancancan qui facilite la gestion des droits, ainsin que la gem rolify qui elle crée un modele de groupe et y attribue les id des users

La "rolification" doit etre présente aussi dans la seed.

Ces deux outils se combinent plutot bien, mais qu' en est il de l'association avec rails admin ?

`Stories:`

- Un administrateur crée en meme temps que la db peut attribuer des roles.
- Deux rôles sont disponibles : "admin" et "mod".
- Un administrateur peux TOUT editer.
- Un modérateur peut lui SEULEMENT éditer les marques et les produits : création, modification et suppression
- Un utilisateur simple peut s'identifier mais n'aura aucun de ces privilèges, à moins qu'un admin ne lui attribue un rôle.
- Aucun "url re-writing" ne permet à quiquonque d'outrepasser les permissions dues

à son role.

Chapitre X

Exercice 06 : Show me what you got

	Exercice : 06
Exercice 06 : Show me what you got	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `acme`	
Fonctions Autorisées : `functions_authorized`	

Mettez en ligne votre application avec un hébergeur de votre choix.

`Stories:`

- Notre communauté de beta testeurs est prête l'application est disponnible sur le web.
- Vous devez avoir votre login visible sur votre site internet.
- Un administrateur peut TOUT éditer.
- Un script de peuplement de l'application permet de remplir l'application avec 2500 produits, 50 marques, 20 users dont un "admin" et 5 "modos"
- Toutes les fonctionalités mise en évidence via les stories du jour sont fonctionelles en ligne : upload d'images, authentification, panier, roles etc.

Chapitre XI

Rendu et peer-évaluation

Rendez votre travail dans votre dépôt `Git` comme d'habitude. Seul le travail présent dans votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

`L'évaluation se déroulera sur l'ordinateur du groupe évalué.`