

# D07 - Formation Ruby on Rails

English version coming soon!

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.

# Table des matières

Ι	Préambule	2
II	Consignes	4
III	Règles spécifiques de la journée	6
IV	Exercice 00 : MySQL is a bad habit	7
$\mathbf{V}$	Exercice 01 : You Sir?	8
VI	Exercice 02 : Get me something to sell	9
VII	Exercice 03 : Panier	11
VIII	Exercice 04: One panel to rule them all	13
IX	Exercice 05 : One account to rule them all	14
$\mathbf{X}$	Exercice 06 : Show me what you got	16

## 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

### Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
  - $\circ$  Ruby >= 2.3.0
  - o pour d09 Rails > 5
  - o mais pour tous les autres jours Rails 4.2.7
  - o HTML 5
  - o CSS 3
- Nous vous <u>interdisons FORMELLEMENT</u> d'utiliser les mots clés while, for, redo, break, retry, loop et until dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre <u>la procédure de rendu</u> pour tous vos exercices : seul le travail présent sur votre dépot GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous <u>ne devez</u> laisser dans votre répertoire <u>aucun</u> autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Vous avez une question? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle Google / man / Internet / ....
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...

- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin! Réfléchissez nom d'une pipe!

## Chapitre III

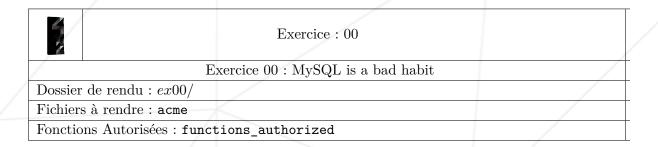
## Règles spécifiques de la journée

- 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



Commençons par un peu d'AdminSys. Installez postgresql sur la machine 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 Heroku

## 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.

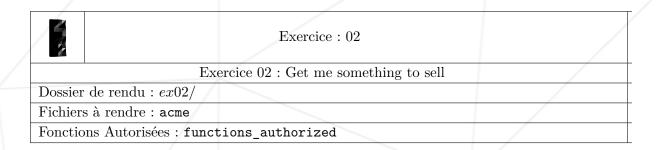
Si vous inspectez le Gemfile, vous verrez qu'elle est de ja présente. Il vous suffit maintenant de l'installer et de la faire gérer un model "User" de sorte que votre seed puisse éxécuter les commandes suivantes :

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

- 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



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 :

- 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.

- 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

1	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.

- 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

1	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.

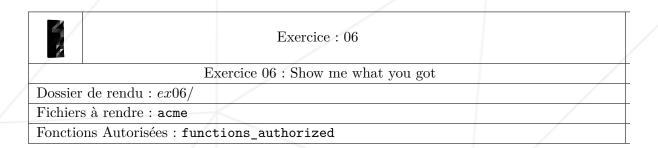
Ces deux outils se combinent plutot bien, mais qu' en est il de l'association avec rails admin?

- 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

	/_	
<u>D07 - For</u>	rmation Ruby on Rails	English version coming soon!
às	son role.	
	15	5

## Chapitre X

## Exercice 06: Show me what you got



Creez un compte sur  ${
m Heroku} < 3$  (oups, ca m'a echappé). Créez un compte disais-je et mettez en ligne votre application.

- Notre communauté de beta testeurs est prête l'application est disponnible sur le web.
- Elle est disponible à l'adresse' : "https://votrelogin-acme.herokuapp.com"
- 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.