



D07 - Formation Ruby on Rails

Mise en situation et pratiques avancées

Staff 42 bocal@staff.42.fr

Stéphane Ballet balletstephane.pro@gmail.com

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

I	Préambule	2
II	Consignes	4
III	Règles spécifiques de la journée	6
IV	Exercice 00 : MySQL is a bad habit	7
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
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 :
 - Ruby `>= 2.3.0`
 - pour d09 Rails `> 5`
 - mais pour tous les autres jours Rails `4.2.7`
 - HTML `5`
 - CSS `3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while`, `for`, `redo`, `break`, `retry` 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 la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépôt GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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 améliorées 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 fonctionnalités présentes. 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 : <i>ex00/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

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 nommée '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'appelle "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 : <i>ex01/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

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

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 déjà présente. Il vous suffit maintenant de l'installer et de la faire gérer un model "User" de sorte que votre seed puisse exécuter les commandes suivantes :


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

Stories:

- Un utilisateur peut créer un compte avec un mot de passe (nécessaire), un nom (nécessaire), un email (nécessaire) et une biographie (optionnelle).
- Il peut re-editer tout ces champs après création 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 : <i>ex02/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

Créez des produits à vendre et des marques, et comme tout produit outre une description très 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 c'est qu'un hébergement gratuit supporte très 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 créer un compte gratuit chez cloudinary, et vous octroyer par là un espace de stockage distant spécialisé dans les images et autres medias.

A ce stade votre seed peut exécuter :

```
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 produits doit afficher les 2500 images, dans une version thumbnail afin de charger rapidement.
- On peut créer et/ou éditer tous les champs en ligne des marques et des produits.
- À terme des rôles seront attribués, déterminant qui peut éditer quoi.

Chapitre VII

Exercice 03 : Panier

	Exercice : 03
Exercice 03 : Panier	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

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 nécessitant pas de CRUD à proprement parler, seul un model leur est utile. Cependant, des fonctionnalités devront être placés dans un "Concern" afin de pouvoir inclure des méthodes relatives au panier dans d'autres contrôleurs 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" présent sur chaque article.
- L'utilisateur peut commencer une commande, remplir son panier et fermer le navigateur. Lors de son retour sur le site son panier est rechargé.
- L'utilisateur peut 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 récapitulatif 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 : <i>ex04/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

Vous devez maintenant créer un panneau 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 possède un compte. Criez sur la page catalogue un link qui mène au dashboard fraîchement disponible.

Stories:

- Un utilisateur enregistré peut 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 : <i>ex05/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

Il est, vous me l'accorderez, plutot inutile de disposer d'un panneau d'administration, si tout le monde a la possibilité 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 cancan qui facilite la gestion des droits, ainsi 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 peut TOUT editer.
- Un modérateur peut lui SEULEMENT editer les marques et les produits : creation, modification et suppression
- Un utilisateur simple peut s'identifier mais n'aura aucun de ces privilèges, a moins qu'un admin ne lui attribue un role.

- Aucun "url re-writing" ne permet a quiconque 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 : <i>ex06/</i>	
Fichiers à rendre : acme	
Fonctions Autorisées : functions_authorized	
Remarques : n/a	

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

Stories:

- Notre communauté de beta testeurs sont prêts l'application est disponible sur le web.
- Elle est disponible @ : "https ://votrelogin-acme.herokuapp.com"
- Un administrateur peut TOUT editer.
- Un script de peuplement de l' application permet de remplir l'application avec 2500 produits, 50 marques, 20 users dont un "admin" et 5 "mod"
- Toutes les fonctionnalités mise en evidence via les stories du jours sont fonctionnelles en ligne, upload d'images, authentication, panier roles etc etc....