



D07 - Ruby on Rails Training

Contextualization and advanced techniques

Summary: A day dedicated to the practical applications of web dev. You will be confronted to a contextualization via 'stories' that will lead to the creation of a basic e-shopping site.

Contents

I	Preamble	2
II	Ocaml piscine, general rules	4
III	Today's specific instructions	6
IV	Exercise 00: MySQL is a bad habit	7
V	Exercise 01: You Sir?	8
VI	Exercise 02: Get me something to sell	9
VII	Exercise 03: Cart	11
VIII	Exercise 04: One panel to rule them all	12
IX	Exercise 05: One account to rule them all	13
X	Exercise 06: Show me what you got	14

Chapter I

Preamble

Write a user story (scenario, story)

A user story is a string of actions performed by someone using our application followed by one or several tests validating the proper conduct of our story.

To be clear, a story looks like a mathematical problem that would include:

- hypothesis
- a disruptive factor
- something to prove

Mathematical problem example (middle school level) :

- There are 2 persons (Peter and John)
- Peter owns 3 bananas
- John owns 2 times more bananas than Peter when John eats one banana
- How many bananas does John own then?

Let's change this example in a mathematical manner (College level, thank you M. Bool):

- There are 2 persons (Peter and John)
- Peter owns 3 bananas
- John owns 2 times more bananas than Peter when John eats one banana
- Let's prove John owns 5 bananas.

If we were to write a user story validated by our system, we would write:

- There are 2 persons (Peter and John)
- Peter owns 3 bananas
- John owns 2 times more bananas than Peter when John eats one banana
- John should own 5 bananas.

This is what we call a test: John **should** own 5 bananas.

If we write tests, this is because our system **must** behave like this.

This helps us validate that our application ALWAYS reacts the same way, even after months or years (hence, several modifications).

Documentation, resources, references:

- [RSpec, cucumber book](#)
- [Le wiki du github de cucumber](#)
- [the cucumber site](#)
- [Motivational](#)

Chapter II

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter III


Today's specific instructions

- All of today's work will be enhanced version of the same Rails application.
- You cannot add anything to the provided Gemfile.
- Any global variable is prohibited.
- You must turn-in a seed in line with the included functionanlites. During the evaluation, your assessor must be able to visualize your work.
- rubycritic must grant you a score of at least 89/100.
- You must manage errors. No Rails error page will be tolerated during the evaluation.

Tips: If you want the evaluation to be simpler, faster and cooler, write tests in connected to the stories! This will make everybody's life easier, and you must get used to doing this. You will need it throughout your career.

Chapter IV

Exercise 00: MySQL is a bad habit

	Exercise 00
Exercise 00: MySQL is a bad habit	
Turn-in directory : <i>ex00/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

Let's start with a little AdminSys. Install `postgresql` on the machine and create a template and a user. Thus, you can create an Rails application names 'acme' using the Gemfile you will find in the `d07.tar.gz` tarball.


You must make sure the "rake db:create" command passes without any error.

Story:

- The application is called "acme". I want to be able to upload it on [Heroku](#)

Chapter V

Exercise 01: You Sir?

	Exercise 01
Exercise 01: You Sir?	
Turn-in directory : <i>ex01/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

You have a DB in a brand new app. Yesterday, you've created an authentication manually, but IRL, that's not the way it's done. Actually, even for a little blog, you have a server that **has** to be protected.

In order to do that, you use a **very** good **currency** library that, besides making breakfast (coffee is just not enough anymore), manages authentications.

If you inspect Gemfile, you will see it's already present. You just have to install it and make it manage a "User" model so that your seed can execute the following commands:


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

Stories:

- A user can create an account with a password (mandatory), a name (mandatory), an email address (mandatory) and a biography (optional).
- They can re-edit all these fields après the account creation via the application (configure your "strong parameters")

Chapter VI

Exercise 02: Get me something to sell

	Exercise 02
Exercise 02: Get me something to sell	
Turn-in directory : <i>ex02/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

Create products and brands to sell. Beyond a luscious description, selling an out of this world quality, you will need an image... this is a problem.

Indeed, in order to properly upload the image file, we're gonna use the **carrierwave** gem (the classier solution). Nothing too complicated up until now. The thing is a free domain hosting doesn't react too well with voluminous data.

This is why the Gemfile includes a gem called Cloudinary. You must create a free account at [cloudinary](https://cloudinary.com), and get yourself a remote storage space specialized in images and other media (in other words, a CDN).

Now, your seed should be able to execute:

```
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:

- Login on the application site, we can see the products' catalog.
- Each product has a name, an image, a description, a brand and a price.
- A brand has a name and an image.
- Whichever the size of the uploaded image, the product page must display the 2500 images as thumbnails so it loads quickly.
- We can create or edit every brand's and product's online fields.
- Roles will be attributed forward. They will determine who can edit what.

Chapter VII

Exercise 03: Cart

	Exercise 03
Exercise 03: Basket	
Turn-in directory : <i>ex03/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

We need to create a **Cart** that will be associated to product copies as **CartItem**, copied in **OrderItem** and assembled in an **Order** when validating the cart.

These specific objects will not need a proper CRUD. They will just need a model. However, functionalities will have to be placed in a "Concern" so they can include methods pertaining to the cart in other controllers such as the "products" one. Create a `current_cart` method based on the `session_id`.


You should also destroy useless objects and registrations. For instance, when canceling a cart, associated objects must be destroyed.

Stories:

- In the catalog page, a 'cart' insert is present.
- You can add items clicking the button "Add to cart" present with each article.
- The user can start an order, fill their cart and close the browser. When they're back, their cart is reloaded.
- The user can increase and reduce the number of items in their cart.
- The cart's lines show an item type, its quantity, a plus button and a minus button, as well as the price according to the formula: quantity * price.
- The user can cancel a cart and empty it with a button.
- A "Checkout" button display an order recap and a total price.

Chapter VIII

Exercise 04: One panel to rule them all

	Exercise 04
Exercise 04: One panel to rule them all	
Turn-in directory : <i>ex04/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

Now, you will create a worthy administration panel.
In the Gemfile, you will find a rails_admin gem. Go find the documentation and initialize it.


(For now) you can access it if you own an account. On the catalog page, create a link that lead to the brand new dashboard.

Stories:

- A registered user can login and access the site's administration panel.
- From there, you can edit and visualize all the site's data.

Chapter IX

Exercise 05: One account to rule them all

	Exercise 05
Exercise 05: One account to rule them all	
Turn-in directory : <i>ex05/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

You will agree on this: it's rather useless to have an administration panel if everyone can dictate their rules as soon as they're registered.

On this occasion, I've included the `cancancan` gem, that helps manage the authorizations, as well as the `rolify` gem, that creates a group model and attributes users' ids.

The "rolification" must also be present in the seed.


Both those tools complement each others well, but who do they fit Rails admin ?

Stories:

- An administrator created together with the db can attribute roles.
- Two roles are available: "admin" and "mod".
- An administrator can edit EVERYTHING.
- A moderator can ONLY edit brands and products: creation, modification, and suppression.
- A simple user can login but they will not access these privileges unless an admin attributed them a role.
- "Url re-writing" will never allow anyone to override the authorization their role grants them.

Chapter X

Exercise 06: Show me what you got

	Exercise 06
Exercise 06: Show me what you got	
Turn-in directory : <i>ex06/</i>	
Files to turn in : acme	
Allowed functions : functions_authorized	

Create an account on [Heroku](#) <3 (sorry about this one). So, well, create an account and upload your application.

Stories:

- Our community of beta testers is ready, the application is now online.
- It is available at the following address: "https://votrelogin-acme.herokuapp.com"
- An administrator can edit EVERYTHING
- An application populating script allows to fill the application with 2500 products, 50 brands, 20 users, among which one "admin" and 5 "mods".
- All the functionalities required by today's stories are implemented and working online: image upload, authentication, cart, roles, etc.