Ruby on Rails Training - 3

Advanced

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.

Version: 1

Contents

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

General rules

- Your project must be realized in a virtual machine.
- Your virtual machine must have all the necessary software to complete your project. These softwares must be configured and installed.
- You can choose the operating system to use for your virtual machine.
- You must be able to use your virtual machine from a cluster computer.
- You must use a shared folder between your virtual machine and your host machine.
- During your evaluations you will use this folder to share with your repository.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

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 : *ex*00/	
Files to turn in : `acme`	
Allowed functions : `functions_authorized`	

Let's start with a little AdminSys. Install `postgresql` on the virtual 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 any provider (online of your choice).

Chapter V

Exercise 01: You Sir?

	Exercise 01
Exercise 01: You Sir?	
Turn-in directory : *ex*01/	
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 : *ex*02/	
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, and get yourself a remote storage space specialized in images and other media (in other words, a CDN).

Ǹow, 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 : *ex*03/	
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 : *ex*04/	
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 : *ex*05/	
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 : *ex*06/	
Files to turn in : `acme`	
Allowed functions : `functions_authorized`	

Create an account on the host of your choice and upload your application.

`Stories:`

- Our community of beta testers is ready, the application is now online.
- You must have your login visible on your website.
- 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.

Chapter XI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

`The evaluation process will happen on the computer of the evaluated group.`