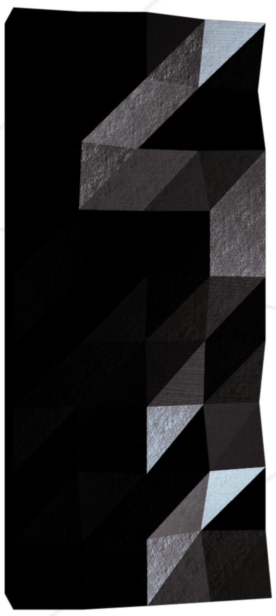

Ruby on Rails Training - 3

Advanced

Summary: A day on applied web developpement practices. You will be put in sceniarios through "stories" that will lead to the creation of a basic e-shopping website.

Version: 1.2

Contents

I Preamble 2

II General rules 4

III Today's specific instructions 5

IV Exercise 00: MySQL is a bad habit 6

V Exercise 01: You Sir? 7

VI Exercise 02: Get me something to sell 8

VII Exercise 03: Cart 10

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

XI Submission and peer-evaluation 15

Chapter I

Preamble

Write a user story (scenario, story)

`A user story is series of actions performed by a user of our application, followed by one or more assertions (tests) validating that our story has proceeded correctly.`

More clearly, a user story resembles a mathematical problem in which we have:

- hypotheses
- a disruptive element
- something to demonstrate

Example of a mathematical problem (middle school level) :

- Let's consider 2 people, Pierre and Jean.
- Pierre has 3 bananas.
- Jean has twice as many bananas as Pierre when Jean eats one banana.
- So, how many bananas does Jean have?

Let's transform this example in college level (tanks to M. Bool):

- Let's consider 2 people, Pierre and Jean.
- Pierre has 3 bananas.
- Jean has twice as many bananas as Pierre when Jean eats one banana.
- So, Let's prove that Jean has 5 bananas.

If now we want to write a user story validated by our system, we will write:

- Let's consider 2 people, Pierre and Jean
- Pierre has 3 bananas
- Jean has twice as many bananas as Pierre when Jean eats one banana
- then Jean should possess 5 bananas.

This is what we call a test: John **should** possess 5 bananas.
If we write tests, it's because our system **must** behave as such.
This allows us to validate that our application ALWAYS reacts in the same way, even after many months/years (after many modifications).

Docs, sources and references:

- Rspec, cucumber book
- Cucumber GitHub wiki
- the cucumber website
- 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

- The final assignment for this module will be a series of improved versions of the same Rails application.
- Any adding to the provided Gemfile is prohibited.
- Usage of any global variable is prohibited.
- You must submit a seed in accordance with the present functionalities. During your evaluation, your evaluator must be able to visualize your work.
- rubycritic must award you a score of at least 89/100.
- You must handle errors. No Rails error page will be tolerated during evaluation.

`Tips:` If you want your evaluation to be simpler, faster and more enjoyable, write tests related to the stories! It simplifies everyone's life and is an essential, no, IMPERATIVE habit to adopt for your future.

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 some AdminSys tasks. Install `PostgreSQL` on your virtual machine and create a template and a user. Then create a Rails application named "acme" using the Gemfile that you'll find in the attachment.

You must ensure that your application uses your PostgreSQL database and that the "rake db:create" command runs without errors.

`Story:`

- The application is named "acme".
- I want to be able to deploy the application to an online host (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 fresh app. In the previous module, you manually created authentication, but in real life, we don't do it that way. Indeed, even for a small blog, a server is present and `must` be protected.

To achieve this, we use the `excellent` library called "devise", which not only makes breakfast (at this point, the coffee is already too far away), but also handles authentications.

If you inspect the Gemfile, you will see that it is already present. Now, all you need to do is 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 (mandatory) and a biography (optional).
- They can edit all these fields after creating the account through 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 for sell and brands, and like any product, besides a highly advantageous description boasting merits that physics refutes, an image is required... and that's our problem.

Indeed, to enable proper file upload,we will use the gem `carrierwave` (the classier solution). Nothing too complicated so far. The catch now is that free hosting does not handle large data storage very well.

For this purpose, there is a gem called "cloudinary" in the Gemfile. You must create a free account at cloudinary, and thereby grant yourself remote storage space specialized in images and other media (a CDN, you know).

At this point, 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:`

- When login into the application's website, users see a catalog listing products.
- Each product consists of a name, an image, a description, a brand and a price.
- A brand has a name and an image.
- Regardless of the uploaded image's size, the product page should display the 2500 images in a thumbnails version for quick loading.
- Roles will be assigned later, determining 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 shopping cart, called `Cart` where copies of products will be stored as “CartItem”.

These “CartItem”, will be copied as “OrderItem” and gathered int an “Order” when the cart is validated.

Since these specific objects do not require a full CRUD. only a model is necessary for them. However, certain functionalities need to be organized in a “Concern” to include cart-related methods in other controllers, such as the “products” controller. You should create a method “current_cart”that relies on the session_id. to identify the current cart

Also, it is essential to remove unnecessary objects and records. For instance, when a cart is canceled, the associated “CartItem” items, must be destroyed to avoid unnecessary data.

`Stories:`

- In the catalog page, there is a “Cart” section.
- Users can add items to the cart by clicking the “Add to cart” button on each product.
- The user can start a purchase, fill their cart and close the browser. When they return to the website, their cart is reloaded.
- The user can increase and decrease the quantity of each item in the cart.

- Cart row display the item type, its quantity, plus and minus button, and the price based on the formula: quantity * price.
- The user can cancel the cart and make it empty with a button.
- A “Checkout” button shows a summary of the order with the 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`	

You must now create a proper administration panel. In the Gemfile, there is a gem called rails_admin. Go check the documentation and set it up.

Currently, access to the admin panel is granted as soon as you have an account. Create a link on the catalog page that leads to the newly available dashboard.

`Stories:`

- A registered user can log in and access the site's administration panel.
- From there, users can edit and view 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`	

It is undeniable that an administration panel would be rather useless if everyone had the posssibility to do as they wish as long as they have registered.

For this occasion, we have included two gem: "cancancan" which facilitates rights management, and "rolify" which creates a group model and assigns user IDs to it.

The "rolification" must also be present in the seed to assign roles during the creation of the database.

These two tools combine quite well, but what about the association with Rails admin?

`Stories:`

- An administrator created at the same time as the database, can assign roles.
- Two roles are available: "administrator" and "moderator".
- An administrator can edit EVERYTHING.
- A moderator can ONLY edit brands and products: creation, modification, and deletion.
- A regular user can log in, but they won't have any of these privileges unless an admin assigns them a role.
- No Url rewriting allows anyone to by pass the permission due to their role.

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`	

Put your application online with a hosting service of your choice.

`Stories:`

- Our community of beta testers is ready, and the application is available on the web.
- Your login should be visible on your website.
- An administrator can edit EVERYTHING
- A seeding script populates the application with 2500 products, 50 brands, 20 users, including 1 “administrator” and 5 “moderator”.
- All the functionalities required in the stories of this module are functional 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.`