



Ruby on Rails Training - 3

Sessions

Summary: You're about to discover the wonderful world of rights through cookies, flags, authorizations and privileges.

By the way, Rails has a very clear error management. Use it. At your level, the 'error-driven development' could be a relevant option!

Version: 1

Contents

I	Preamble	2
II	General rules	3
III	Today's specific instructions	4
IV	Exercise 00: It's me	5
V	Exercise 01: Add me in	7
VI	Exercise 02: Need an account	9
VII	Exercise 03: Peer edit	11
VIII	Exercise 04: UvDv	12
IX	Exercise 05: Can you?	13
X	Submission and peer-evaluation	14

Chapter I

Preamble

And talking about authorizations and privileges, here is a little note from M. Gates. It's a little outdated, but if you put history in perspective, it's rather funny.

-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds \$40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than \$2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates
Bill Gates
General Partner, Micro-Soft

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


- Today's work will be different enhanced versions of the same Rails application.
- You cannot add anything to the Gemfile provided in the resources.
- Global variables are prohibited.
- You must provide a seed that you will implement through today's exercises. During the evaluation, no exercise will be valid if the implemented functionalities are not represented in the db's data.
- You MUST manage errors. No Rails error page will be tolerated during the evaluation.
- You must have a score awarded by rubycritic superior or equal to 90.
- Rails_best_practice must not report ANY 'warning'.
- You must NOT touch the "rails_best_practices.yml" file and place it at the root of "app/config/". (it contains a soft config for the d06)



If you want the evaluation to be faster, cooler and easier, write tests! It will make everyone's life easier and you should develop the habit do to them. Well, you actually MUST develop that habit, because it will be necessary everyday of your working life.

Chapter IV

Exercise 00: It's me

	Exercise 00
Exercise 00: It's me	
Turn-in directory : <i>ex00/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

You must create a brand new app. It will get more and more complex through the day, so take a good start: code properly and use git.

Where was I... Yes ! Authentication. You must create a decent one, with encryption, redirection and all. But in order to make it good, we're gonna have to dissect the exercise as should be.

For the view, just create a 'root page' for now. It will include:

- A header named "User Line". Once the user is logged, it will have the following form:

```
Welcome <UserName> !   Log_out
```

- If the user is anonymous, the User Line will look as follows:

```
Welcome <RandomAnimalName>_visitor ! Sign_in Log_in\
```

- "Sign_in" will send a simple registration form that will include:
 - A name
 - An mail address
 - A password
 - A password_confirmation
- The registration form doesn't show the passwords.

- "Log_in" will redirect you to a classic login form with:
 - A field taking in account the name or the mail address (or both if you feel brave)
 - The password (obviously, it will still not show)
- "Log_out" will... log you out.

The code to get the <RandomAnimalName> is provided:

```
url = "https://www.randomlists.com/random-animals"
doc = Nokogiri::HTML(open(url))
name = doc.css('li').first.css('span').text
```

You should work things out to properly implement this snippet.

For the controller and model, you will integrate the following elements:

- A user must have a name, a mail address and a password.
- Users must have a UNIQUE and a UNIQUE mail address.
- An anonymous user will have an animal name stored in a cookie with 1 minute validity (easier for evaluation). Thus, after 1 minute, their name will change for the following one.
- After registering, the new user will be automatically logged.
- Passwords don't appear clearly in the server log and they're not stored in the base.
- The password will have at least 8 characters.
- The form will display errors with a message, not with the debug console.


You will also create a seed with a few users, just to make sure your model works. You will also check that your controller does log you test accounts. Moreover, you really **should** have tests.

The 'rails_best_practice' utility must not return any warning:

```
%>rails_best_practices
Source Code: |=====|
Please go to http://rails-bestpractices.com to see more Rails Best Practices.
No warning found. Cool!
```

Chapter V

Exercise 01: Add me in

	Exercise 01
Exercise 01: Add me in	
Turn-in directory : <i>ex01/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

Ok great! Now we own a user DB and an authentication. This is basic, but it's just a beginning.

You will add an **admin** section to administrate users. For further security, it will lie on a pattern design that uses the namespacing.

You will broadly have to make things so the the URL about the actions requiring admin rights start with `"/admin/..."`. Thus, you will be able to implement a controller that concerns the users, without any conflict with the admin controller.

The new controller will have to get prototyped as follows:

```
class Admin::UsersController < ApplicationController

  #=====
  # your code here
  #=====

end
```

So the `'http://localhost:3000/admin/users'` URL is available and leads to the created page.

I truly hope you have used the correct conventions and that `current_user` matches the logged user or the cookie's one.

Administrators must be differentiated by an `admin` boolean (set on false by default) type field in the users' table.

They will be the only ones who can access the list of every user, edit values (names and mail address) and delete them.

However, each user can change **their own** names and emails.


When a user is admin when they login, their message is followed by a link that gets them to the 'admin/users' page (the index of our new controller):

```
Welcome <AdminName> ! Administrate_users Log_out
```

Besides 'callbacks', you will have to properly configure [routes](#).

Chapter VI

Exercise 02: Need an account

	Exercise 02
Exercise 02: Need an account	
Turn-in directory : <i>ex02/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

The back-office base being almost over, we're gonna add a little meaning to our app. Indeed, the LifeProTips application is meant to receive and share "life pro tips". But in order to make it a useful tips database, we're gonna have to restrain the access to members. But not too much...

With the help of a scaffold, you must create a Post object that will contain:

- a 'user_id' (mandatory)
- a 'title' (unique, 3 char minimum)
- a 'content' (that can contain a bulk of text)

Of course, the author will not set their id themselves when creating content. It is automatically made with the `current_user` within the controller.

The index must list titles and authors by name and sorted out by date in decreasing order (youngest first).

Visitors can only see the index. The title must be a link to the 'post show' action and if a visitor clicks it, they will be redirected to the 'root page' with the following notice: "You need an account to see this".

The new 'root page' is the posts index.

A user must be able to edit all the posts (for now).

Repeat the same operation as in exercise 01: an admin page for the posts or an administrator has all the rights.


The header line becomes (if admin):

```
Welcome <AdminName> ! Administrate_users Administrate_posts Log_out
```

The "Administrate_users" link leads to a page listing all the created posts. It give access to their removal and modification.

Chapter VII

Exercise 03: Peer edit

	Exercise 03
Exercise 03: Peer edit	
Turn-in directory : <i>ex03/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

Unwise the one who thinks he knows all.

The Life Pro Tips are editable. Each show page must display who modified the post last and when:

```
Original author:  bob2
Title:  How do you know water is too hot?
Content:  Dip your baby inside and check temperature with your wrist
Edited by:  Stephan
Date of mofication:  2022-03-03 23:42:00 UTC
```


You must implement the functionality represented by the last two lines in the example.

You're free to choose your solution. There are many available, but the good ones are not that usual..

WARNING: neither global variables nor cookie based solutions.

Chapter VIII

Exercise 04: UvDv

	Exercise 04
Exercise 04: UvDv	
Turn-in directory : <i>ex04/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

On the "post show" page, add the option to 'Upvote' and 'Downvote', as well as show the total votes (even if negative).

Vote redirects to the same page.

Nb: no global variables or affiliated solutions. The votes `texttt` must be an object inheriting `ActiveRecord`.


Once again, you must have an admin interface like in exercise 01 (with the same design).

This interface will include a view that lists votes grouped by posts indicating the post title in the header and names of the associated voters.

Administrator can suppress votes.

Chapter IX

Exercise 05: Can you?

	Exercise 05
Exercise 05: Can you?	
Turn-in directory : <i>ex05/</i>	
Files to turn in : LifeProTips	
Allowed functions :	

For today's last exercise, you will have to set privileges based on the number of votes.

- 0 to 3: no specific right.
- 3 to 6: right to upvote.
- 6 to 9: right to downvote.
- above 9: right to edit posts.

Now, a user can only edit their own posts, except if they have more than 9 upvotes.

Users' detail page (user show) shows the rights of the user. These rights also appear on the the users administration page.

Chapter X

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.