Ruby on Rails Training - 3

Sessions

Summary: You are about to discover the wonderful world of rights, through cookies, flags, permission and privileges.
By the way, Rails has a very clear error handling systeme, so make use of it. At this stage of your lerning, "error-driven development" is a good option!

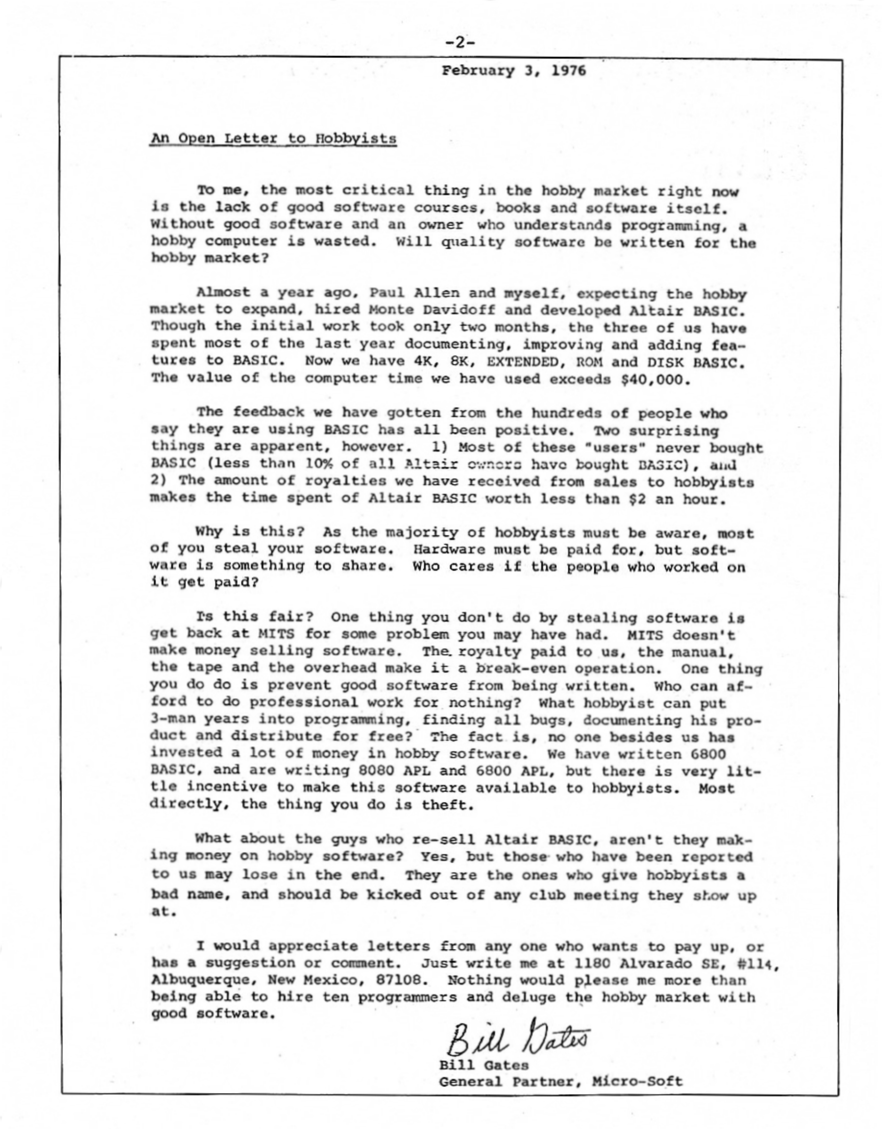
Version: 2

Contents

Chapter I

Preamble

And speaking of permissions and privileges, here's a little letter from M. Gates. That's a bit old but rather amusing in light of history. rather funny.



-2-

February 3, 1976

<u>An Open Letter to Hobbyists</u>

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

- All the work for this module will be improved version of the same Rails application.
- Any addition to the Gemfile provided in the resources is prohibited.
- Any global variables is prohibited.
- You must provide a seed, which you will implement throughout the exercises in this module. During the evaluation, no exercise will be considered valid if the implemented functionalities are not represented in the database data
- You MUST handle errors effectively. No Rails error pages will be tolerated during the evaluation.
- You must have a RubyCritic score greater or equal to 90.
- You must have NO warning reported by Rails_best_practice.
- You must LEAVE the "rails_best_practices.yml" file intact and place it at the root of "app/config/"(it contains a relaxed configuration for this module06).

`If you want your evaluation to be easier, faster and more enjoyable, write tests! It simplifies life for everyone and is an essential, I dare say, IMPERATIVE habit to adopt for your future as a developer.`

Chapter IV

Exercise 00: It's me

	Exercise 00
Exercise 00: It's me	
Turn-in directory : *ex*00/	
Files to turn in : `LifeProTips`	
Allowed functions :	

You must create a brand new app. It will get more complex throughout the day, so please... for your own sake, code properly and use Git. If your authentication has been coded with anything other than your hands... it will bother you all day long...

Now, where are we... Ah Yes ! Authentication. You should create a decent authentication system,complete with encryption, redirection and everything that's needed. But, to do all this properly, we will break down the exercise step by step.

For the view, create a "root page" for now that contains:

- A header named "User Line" which once the user is logged in, takes the following form:

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

- If the user is anonymous, "User line" takes the form:

```
        Welcome <RandomAnimalName> visitor! Sign\_in Log\_in\\
```

- "Sign_in" should redirect the user to a simple registration form that asks for the following information:
    - A name
    - An mail address
    - A password
    - A password_confirmation

- The registration form does not display the passwords in plain text.
- "Log_in" should redirect the user to a standard login form that ask for the following information:
  - The user's name or email (both if you can handle it).
  - The password (which, of course, not appears in plain text)
- "Log_out" allows the user to log out.

To get the <RandomAnimalName>, you can use the following code:

```
animal_names = ['Lion', 'Tiger', 'Bear', 'Leopard', 'Elephant', 'Giraffe', 'Zebra', 'Kangaroo']
random_animal_name = animal_names.sample
```

Regarding the controller and model, you must integrate the following elements:

- A user must have a name, an email and a password.
- Users must have a UNIQUE name and a UNIQUE email.
- The anonymous user has their animal name stored in a cookie with a validity of 1 minute (easier for evaluation). Thus, after 1 minute, their name is replaced with a new random animal name upon the next load.
- After registration, the new user will be automatically logged in.
- Passwords do not appear in plain text in the server log and are not stored in plain text in the database.
- The password to be entered must have at least 8 characters.
- The form display errors with a message, not with the debug console.

You must also create a seed with a few users, to see that your model is well done and also to ensure that your controller logs in your test accounts correctly. You should definitely have tests as well.

The "rails_best_practice" utility should not return any warnings:

```
%>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 : *ex*01/	
Files to turn in : `LifeProTips`	
Allowed functions :	

Ok great! We now have a database of users and authentication, which is a good starts.

You need to add an `admin` section to administer users. For security purpose, it should be base on a design pattern that uses namespacing.

Ensure that the URL related to admin actions require admin rights and begin with "/admin/...". Therefore, you must implement a controller that deals with users without conflicts with the admin controller.

The new controller should be prototyped as follows:

```
class Admin::UsersController < ApplicationController

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

end
```

This way, the URL "http://localhost:3000/admin/users" is available and leads to the created page.

If you have followed the correct conventions, the "current_user" corresponds to the logged-in user or, if not, the user from the cookie.

Administrators must be differentiated by a boolean field `admin` (which defaults to false) in the users' table.
Only administrators can access the list of all users, edit or delete their information (names and email).

Each user should be able to modify `their own` information (names and email).

When an administrator logs in, their message is followed by an additional link that takes them to the page "admin/users", the index of our new controller:

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

In addition to the "callbacks", you must configure your routes properly.

Chapter VI

Exercise 02: Need an account

	Exercise 02
Exercise 02: Need an account	
Turn-in directory : *ex*02/	
Files to turn in : `LifeProTips`	
Allowed functions :	

The back-office foundation being almost complete, you now need to add some meaning to this app. The LifeProTips application is intended to receive and share “life pro tips”, but to keep it advantageous, you must restrict access to members, but not too much...

You must create an object called “Post” using a scaffold that contains the following fields:

- A “user_id” (mandatory).
- A “title” (unique at least 3 characters long).
- A “content” (which can contain a substantial amount of text).

Of course, the author does not set their own ID during creation; It is automatically set in the controller using current_user.

The index should list the titles and authors by their names, sorted by descending date (most recent first).

Only the index is visible to visitors, and title should be link to the “post show” action. If a visitor clicks on it, they should be redirected to the “root page” with the notice: “You need an account to see this”.

The new “root page” is the index of the posts.

For now, non-admin users can only edit their own posts.

Repeat the same operation as in ex01: create an admin page for posts where an administrator has all rights.

The header line becomes (if the user is an admin):

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

The “Administrate_posts” link leads to a page listing all created posts. This page allows access to their deletion and modification.

Chapter VII

Exercise 03: Peer edit

	Exercise 03
Exercise 03: Peer edit	
Turn-in directory : *ex*03/	
Files to turn in : `LifeProTips`	
Allowed functions :	

Poorly informed is the one who thinks he knows everything.

As Life Pro Tips are editable, yo must display on the show page of each post whom and when the post was last modified:

```
Original author:  bob2
Title:  How to tell if the water is too hot?
Content:  Dip your baby in it and check its temperature with the inside of
your wrist
Edited by:  Stephan
Date of modification:  2022-03-03 23:42:00 UTC
```

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

You are free to choose your solution. There are many, but the good ones are rarer...

Please note, no global variables or cookie-based solutions are allowed.

Chapter VIII

Exercise 04: UvDv

	Exercise 04
Exercise 04: UvDv	
Turn-in directory : *ex*04/	
Files to turn in : `LifeProTips`	
Allowed functions :	

Add the ability to "Up-vote" and "Down-vote" on the "post show" page, along with displaying the total votes, which can be negative.

Voting should redirect to the same page.

`Do not use global variables or similar techniques.Votes should be represented as an object inheriting from ActiveRecord.`

Of course, an individual user should not be able to vote more than once on the same post. They can only change their vote by +1 or -1.

Once again, you should have an admin interface similar to ex01, with the same design.

This interface should have a view that lists the votes, grouped by posts, displaying the post title as the header and the names of the voters associated with each post.

The administrator should be able to delete votes.

`To facilitate the evaluation of the next exercise, an administrator should be able to upvote multiple times. However, in reality, this will never be the case.`

Chapter IX

Exercise 05: Can you?

	Exercise 05
Exercise 05: Can you?	
Turn-in directory : *ex*05/	
Files to turn in : `LifeProTips`	
Allowed functions :	

For the last exercise of this module, you need to implement privileges based on a voting points system obtained by the user.

The calculation of the points is simple: you need take the difference between the up-votes and the down-votes received by the user across all their posts.

For example, if a user has received 7 up-votes and 4 down-votes for all of his posts, his total voting points will be 3 (the number of voting points can be negative).

Users can then obtain specific privileges based on the number of voting points they have achieved:

- Up to 1 points: no specific rights.
- From 2 to 3 points: right to up-vote posts from other users.
- From 4 to 5 points: right to down-vote posts from other users.
- Beyond 5 points: right to edit posts from other users.

Now, a user who has 10 points or more will have the right to edit posts from other users.

The User detail page (user show) displays the user's specific rights, and these rights are also shown on the users' administration page.

Chapter X

Submission and peer-evaluation

Submit your work to your `Git` repository as usual. Only the work present in your repository will be evaluated during the defense. Make sure that the names of your folders and files are in accordance with the requirement of the subject.

`The evaluation will take place on the evaluated group's computer.`