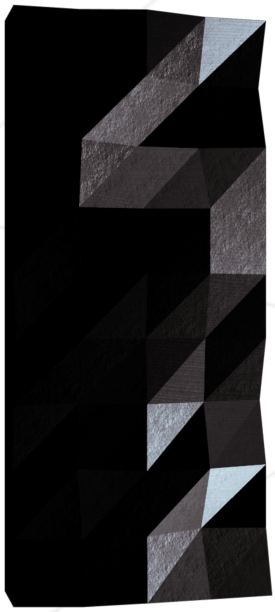


D06 - Formation Ruby on Rails

English version coming soon !

Résumé: Vous allez decouvrir le monde merveilleux des droits , au travers de cookies, de flags, de permissions et privilèges .
Au passage , Rails a une gestion d'erreur très claire, utilisez la. À ce stade de votre apprentissage, le 'error-driven development' est une bonne option !

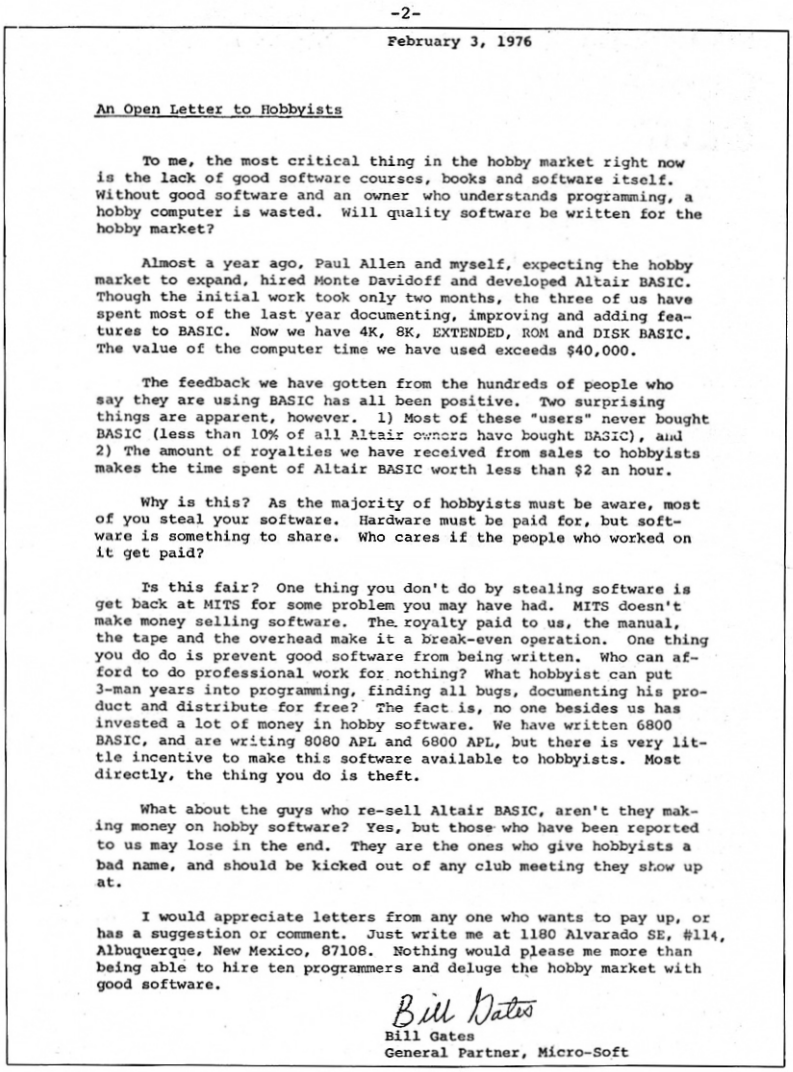
Table des matières

I	**Préambule**	**2**
II	**Consignes**	**3**
III	**Règles spécifiques de la journée**	**5**
IV	**Exercise 00 : It's me**	**6**
V	**Exercise 01 : Add me in**	**8**
VI	**Exercise 02 : Need an account**	**10**
VII	**Exercise 03 : Peer edit**	**12**
VIII	**Exercise 04 : UvDv**	**13**
IX	**Exercise 05 : Can you ?**	**14**

Chapitre I

Préambule

Et en parlant de permissions et de privilèges, une petite lettre de Mr Gates qui date un peu mais qui est plutot drole au vue de l'histoire .



-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

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
  - `Ruby >= 2.3.0`
  - pour `d09 Rails > 5`
  - mais pour tous les autres jours `Rails 4.2.7`
  - `HTML 5`
  - `CSS 3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while, for, redo, break, retry, loop et until` dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépot GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...

- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Tout le travail de la journée sera des versions ameliorées de la même application rails.
- Toute addition au Gemfile fourni dans les ressources est interdite.
- Toute variable globale est interdite.
- Vous devez fournir une seed, que vous allez implementer au fil des exercices de la journée. En correction, aucun exercice ne sera valide si les fonctionalitées implementées ne sont pas representées dans les données de la db.
- vous devez impérativement gérer les erreurs. Aucune page d'erreur rails ne sera tolérée en correction.
- Vous devez avoir un score décerné par rubycritic supérieur ou egal à 90
- Vous devez avoir AUCUN 'warning' signalé par rails_best_practice
- Vous devez LAISSER intacte le fichier "rails_best_practices.yml" et le placer a la racine de "app/config/" (il contient une config adoucie pour le d06)

Si vous voulez que vos corrections soient plus simples, rapides et sympas, écrivez des tests ! Ca simplifie la vie à tout le monde et c'est une habitude primordiale, que dis-je, IMPERATIVE à prendre pour votre avenir.

Chapitre IV

Exercise 00 : It's me

	Exercice : 00
Exercise 00 : It's me	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

Vous devez créer une app toute neuve. Elle va se complexifier toute la journée, donc par pitié (pour vous), codez proprement et utilisez git. Si votre authentification a été codée avec votre assise, elle va vous enquiquiner toute la journée...

Ou en etais-je... Ah oui ! L'authentification. Vous devez en créer une décente, avec encryption, redirection et tout ce qui va bien. Mais pour faire tout cela bien, on va décortiquer l'exercice comme il se doit.

Pour la vue, créez pour l'instant une 'root page'. Elle contiendra :

- Une entête qu'on appellera "User Line" qui, une fois le user logué, aura pour forme

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

- Cet User Line, si le visiteur est anonyme, sera de la forme :

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

- "Sign_in" vous enverra sur un formulaire d'inscription sommaire qui comprendra :
  - Un nom
  - Un email
  - Un password
  - Un password_confirmation
- Le formulaire d'inscription ne laisse pas les passwords apparents.

- "Log_in" vous enverra sur un formulaire de connexion classique avec :
  - Un champ prenant en compte le nom ou l'email (ou les deux si vous êtes chaud)
  - ZE password (qui bien sur ne sera pas apparent non plus)
- "Log_out" vous deconnecte.

Le code pour avoir le <RandomAnimalName> vous est donné :

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

A vous de vous organiser pour implémenter correctement ce snippet.

Côté controller et model, vous devrez intégrer les élements suivants :

- Un User doit avoir un nom, un email, et un mot de passe.
- Les Users doivent avoir un nom UNIQUE et un email UNIQUE.
- L'utilisateur anonyme a son nom d'animal stocké dans un cookie avec une minute de validité (plus facile pour les corrections). Ainsi, au bout d'une minute, son nom aura changé au chargement suivant.
- Apres son inscription, le nouvel utilisateur est automatiquement logué.
- Les mots de passe n'apparaissent pas en clair dans le log serveur non plus et n'est pas stocké en clair dans la base.
- Le mot de passe à renseigner devra avoir au moins 8 caractères
- Le formulaire affichera les erreurs avec un message, et non avec la console de debug.

Vous devez créer aussi créer une seed avec quelques utilisateurs, histoire de voir que votre model est bien fait, mais aussi que votre controller logue bien vos comptes tests. Vous `devriez` d'ailleurs avoir des tests, vraiment.

L'utilitaire 'rails_best_practice' ne doit renvoyer aucun warning :

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

Chapitre V

Exercise 01 : Add me in

	Exercice : 01
Exercise 01 : Add me in	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

Ok great ! Nous disposons d'une DB d'utilisateurs et d'une authentification. c'est sommaire, mais c'est un début .

Vous allez ajouter une section `admin` afin d'administrer les utilisateurs. Pour plus de securité elle va reposer sur un design pattern qui utilise le namespacing.

En gros : faites en sorte que les url concernant les actions requierant des droits d'admins commencent par "/admin/...". Ainsi, vous pourrez implémenter un controlleur qui concerne les users, sans conflit avec le controlleur admin.

Le nouveau controlleur devra se prototyper comme ceci :

```
class Admin::UsersController < ApplicationController

  #================
  # votre code ici
  #================

end
```

Ainsi, l'url 'http ://localhost :3000/admin/users' est disponible et mène a la page créee.

J'espère pour vous que vous avez utilisé les bonnes conventions et que donc pour vous, current_user correspond à l'utilisateur logué ou sinon à celui du cookie .

Les administrateurs doivent être différenciés par un champ `admin` de type booléen (par defaut à false) dans la table des utilisateurs.
Eux seuls peuvent accéder à la liste de tous les utilisateurs, en éditer les valeurs (nom et email) et les supprimer.

Chaque utilisateur peut cependant changer `son propre` nom ou email.

Lorsqu'un utilisateur est admin à sa connexion, son message est suivi d'un link en plus qui mene a la page 'admin/users' (l'index de notre nouveau controller) :

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

En plus des 'callbacks', vous aurez besoin de bien configurer vos routes.

Chapitre VI

Exercise 02 : Need an account

	Exercice : 02
Exercise 02 : Need an account	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

La base back-office étant quasiment finie, on va ajouter un peu de sens à notre app. En effet, l'application LifeProTips est destinée a recevoir et partager des "life pro tips" mais pour que cela reste un peu des astuces avantageuses nous allons restreindre l'accès aux membres, mais pas trop ...

Vous devez créer à l'aide d'un scaffold un objet Post qui contient :

- un 'user_id' (obligatoire)
- un 'title' (unique et de longueure min de 3 char)
- un 'content' (qui doit pouvoir contenir pas mal de texte)

Evidemment, l'auteur ne met pas son id lui-même à la creation, c'est dans le controlleur que la chose se fait automatiquement avec current_user

L'index doit lister les titles et les auteurs par leur nom et le tout trié date decroissante (le plus jeune en premier).

Seul l'index est visible par les visiteurs, le title doit être un link vers l'action 'post show' et si un visiteur clique dessus, il doit être redirigé vers la 'root page' avec comme notice : "You need an account to see this"

La nouvelle 'root page' est l'index des posts

Un utilisateur doit pouvoir éditer tous les posts (pour l'instant).

Répétez la même operation que dans l'exercice 01 : une page admin pour les posts ou un administrateur a tous les droits.

La ligne en en-tête devient (si on est admin) :

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

Le lien "Administrate_users" mène à une page listant tous les posts crées. Celle ci permet d'accéder à leur suppression et modification .

Chapitre VII

Exercise 03 : Peer edit

	Exercice : 03
Exercise 03 : Peer edit	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

Bien mal renseigné celui qui croit tout savoir.

Les Life Pro Tips étant éditables, vous devez afficher sur la page show de chaque post par qui et quand le post a été modifié la derniere fois :

```
   Original author: bob2
Title: Comment deviner si l'eau est trop chaude?
Content: Trempez votre bébé dedans et verifiez sa temperature avec l'interieur
de votre poignet
Edited by: Stéphane
Date of mofication: 2016-07-19 13:55:51 UTC
```

Vous devez implémenter la fonctionalité qui est representée par les deux dernières lignes de l'exemple.

Vous êtes libre de choisir votre solution. Il y en a beaucoup mais les bonnes sont plus rares..

Attention pas de variables globales, ni de solutions à base de cookie.

Chapitre VIII

Exercise 04 : UvDv

	Exercice : 04
Exercise 04 : UvDv	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

Ajoutez sur la page de "post show", la possibilité de 'Upvote' et 'Downvote', ainsi qu'un affichage du total de votes (pouvant être negatif).

Voter redirige sur la même page.

`Nb:` pas de variable globales ou autre de ce style. Les votes `doivent` être un object heritant d'ActiveRecord .

Encore une fois vus devez avoir une interface admin comme pour l'exerecie 01 (avec le même design).

Cette interface disposera d'une vue qui liste les votes groupés par posts en indiquant le titre du post en en-tête et le noms des votants associés.

L'administrateur pourra supprimer des votes .

Chapitre IX

Exercise 05 : Can you ?

	Exercice : 05
Exercise 05 : Can you ?	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `LifeProTips`	
Fonctions Autorisées :	

Vous devez pour ce dernier exercice de la journée, mettre en place des privilèges basés sur le total des votes.

- 0 a 3 : aucun droit spécifique
- 3 a 6 : droit à l'upvote
- 6 a 9 : droit au downvote
- au dessus de 9 droit d'éditer les posts

Maintenant, un user ne peut editer que ses posts, sauf s'il a au dessus de 9 upvote.

La page de detail des Users (user show) affiche de quel droit dispose l'utilisateur en question, et ces droits figurent aussi sur la page d'administration des utilisateurs.