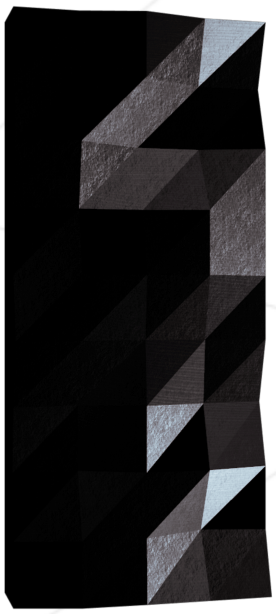


Ruby on Rails Training - 3

Final

Summary: In order to make your pages more interactive and improve the user experience, you need to work on the client-side, which does not know Ruby. Therefore, you are going to use JavaScript.
The first part of the day is dedicated to making the **`CRUD`** *more dynamic using* **`AJAX`**. *For the second part, you are working on real-time multi-client functionality with WebSockets made available to you through Action Cable (a feature introduced since Rails 5).*

Version: 1.1

Contents

Chapter I

Preamble

Since we're going to deal with real time and JavaScript, I've made a list of games in JavaScript:

- Wipeout-like
- Text based rpg
- the ultimate mouse killer
- Puzzle game
- co cow coW cOW COW

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

It's mandatory to use `rails >=5` throughout the this module. Gems that facilitate (or rather do it for you) AJAX are prohibited.

Non-exhaustive list:

- "best_in_place"
- "better-edit-in-place"
- "super_inplace_controls"
- "rest_in_place"
- "on_the_spot"
- "edit_mode"
- "best_in_placeish"
- "crest_in_place"
- ...

You must justify any addition of non-rails-built-in Gems during the evaluation if your assessor ask for it.

`NO` additional JavaScript library is allowed. Your `application.js` should only contain the following imports:

```
jquery
jquery\_ujs
turbolinks
```

Chapter IV

Francis_1

	Exercise 00
Exercise 00:Francis_1	
Turn-in directory : *ex*00/	
Files to turn in : `Xnote`	
Allowed functions :	

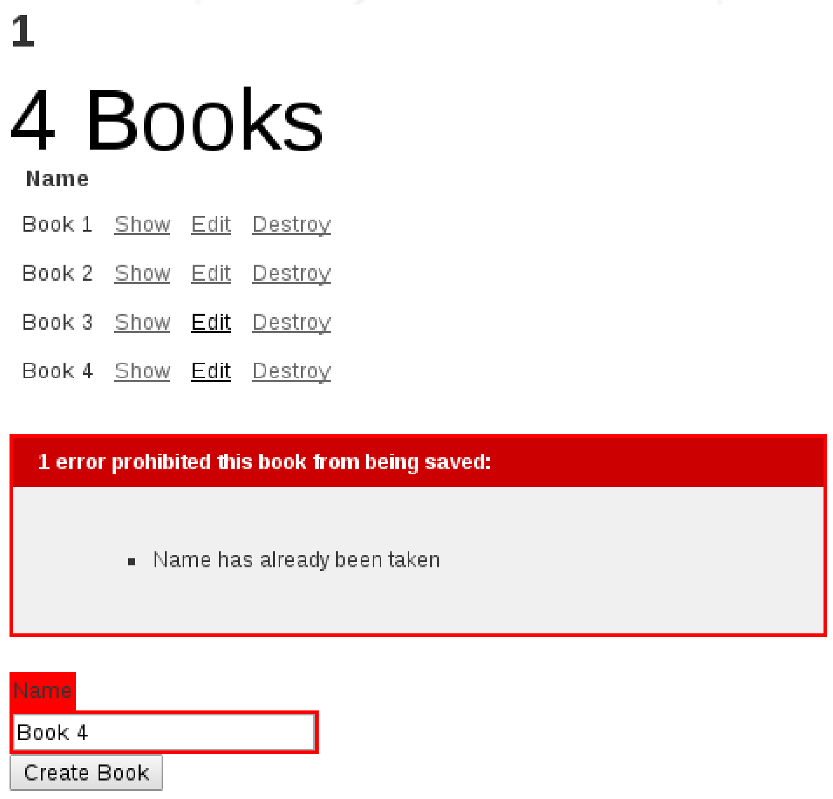
You must create a library listing books. The application is named “Xnote”. For this exercise, a scaffold is sufficient:

```
rails g scaffold book name
```

Then you must:

- Create a uniqueness validation rule for the “name” of “book”.
- Allow the addition of books using AJAX:
  - The form should appear when clicking on the “link_to” that points to “new_book_path”.
  - Update data in the list of “books” upon submitting the form (the form should then disappear).
- display errors, for example, if you try to use a name that is already taken.

Example :



1

4 Books

Name			
Book 1	Show	Edit	Destroy
Book 2	Show	Edit	Destroy
Book 3	Show	Edit	Destroy
Book 4	Show	Edit	Destroy

1 error prohibited this book from being saved:

- Name has already been taken

And all of this `without` refreshing the entire page. To verify this, you must add the following code in your layout:

```
##ex00/Xnote/app/views/layout/application.html.erb:
...
<body>
    <% $refresh ||= 0 %>
    <h1><%= $refresh +=1 %></h1>
    <%= yield %>
</body>
...
```

In this way, your page will enriched with a “refresh count” which should remain at `1`.

Chapter V

Francis_2

	Exercise 01
Exercise 01:Francis_2	
Turn-in directory : *ex*01/	
Files to turn in : `Xnote`	
Allowed functions :	

Now that you understand the principle, it will be easy for you to do the same with a "link_to" pointing to the destroy method. Clicking on it should, after the confirmation popup, remove the entry from the database and update the list.

`The global variable $refresh must remain at 1!`

Chapter VI

Francis_3

	Exercise 02
Exercise 02:Francis_3	
Turn-in directory : *ex*02/	
Files to turn in : `Xnote`	
Allowed functions :	

Once again, it's time for the "edit" method to work without page reloading.

The form should appear in place of the table row corresponding to the "book" being edited while displaying validation errors. For instance, trying to edit a book with an existing name should display a validation error.

`The global variable $refresh must still remain at 1!`

Chapter VII

Francis_4

	Exercise 03
Exercise 03:Francis_4	
Turn-in directory : *ex*03/	
Files to turn in : `Xnote`	
Allowed functions :	

Let move away from the CRUD for this exercise.
At the top of the page, create a count of the total number of books. Is should be updated every time there is modification, whether it's adding a book or deleting one.

`The global variable $refresh must always remain at 1!`

Chapter VIII

ChatOne

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

Until now, we have only produced parts in AJAX, which is just a pattern. Now, let's move towards multi-client functionality.

Create a chat application that includes messages from authenticated users using the "devise" gem. These massages should appear in real-time for all connected users.Name you application "Chat".

A piece of advice: ActionCable handle this very well, so make use of it!

`Open multiple windows in incognito mode and authenticate with different logins.`

Design this application with the intention of handling `a lot` of real-time traffic. You `must` implement a task buffering system.

`the 'ApplicationJob' or 'Active Job' are not just made for our dear canine friends.`

Chapter IX

ChatTwo

	Exercise 05
Exercise 05:ChatTwo	
Turn-in directory : *ex*05/	
Files to turn in : `Chat`	
Allowed functions :	

By reusing the base of the previously created application, implement the concept of ChatRoom: rooms where what is said remains there.

You should ensure that as soon as a user is authenticated, they can create a ChatRoom.

This user is considered the sole and unique creator of this ChatRoom. Deleting this user also deletes all the rooms they have created and all the messages in them. The messages posted only appear in the room were they originally created.

Chapter X

ChatThree

	Exercise 06
Exercise 06:ChatThree	
Turn-in directory : *ex*06/	
Files to turn in : `Chat`	
Allowed functions :	

Always in the same "Chat" application, create a notification system represented in `Chat/apps/views/layouts/application.html.erb` by a list where an entry is added for each new message **if** the author of the message is different from the logged-in user. This makes sense: you don't see notifications for your own messages. This should be applied to all the ChatRooms to which the user is connected.

You must display in real-time, continuously, and on all page, a list of notifications as well as their total count. `NONE` of this notification should be triggered by your own messages. This must work in multi-client and simultaneously.

And while you're at it, add a sound to these notifications as well.

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.