Name: Mathias Klippinge
E-Mail: mathias.klippinge@gmail.com
Phone: 0739 858888
Github: https://github.com/parazyte/pilot

Documentation – Pilot app

This Rails application is a solution to the problem of putting together an external application that displays content served by a CMS API, under the assumption that the "structure of the CMS and how to treat it" is known.

Justification of the development process, strategies, architecture you followed and chose

My interpretation of the problem is:

1) The CMS is an internal server application. It has a well-documented API. We have full control over its source code (in case there is something wrong with it). I can be accessed via GET requests.

2) The application should be built with focus on simplicity and maintainability, it should have structured and readable code and plenty of tests to prove that it works properly.

3) It is also desired that the design of the application should be stable under modification to the content of the JSON response (i.e. more or less games).

With this in mind I started to browse through the server code. The challenge is clearly in how to handle a dynamic change in the index response. Since the CMS will list my games, and also have all information about the games, I came to realize an important design decision: There is no need to have any model(s) related to "games" in this application, since it will not have any game related duties except to display them.

Possible solutions I did consider:

Option 1:

> Make an index method in a regular Rails controller that makes a request to the CMS server to fetch the index from the CMS server and stores its contents in a couple of variables, rendering an index page which would look at these variables.

Pros:

- Simple
- Less JavaScript

Cons:

- Worse user experience since we have to wait for CMS response in order to render page (which in the worst case leaves them with only a blank page and a spinning mouse icon to look at).
- Very questionable choice for requirement [1]! It does not feel like an ok decision to initiate a GET request from a controller, I do not wish to base my design on that.
- Good choice for requirement [2] since we are leveraging off the Rails way and we will also be able to test the application entirely within the familiar domain of Rspec
- Excellent choice for requirement [3] since the templating will keep this in mind

Option 2:

> Make an index method in a regular Rails controller that simply serves a more or less empty file with room for content to be added dynamically via JavaScript.

Pros:

- Better user experience as the page will appear to be loaded and content will arrive later into the assigned viewport when the CMS has responded (asynchronously).
- Seems legit, this is basically how it should be done (at least how I have been taught).
- Excellent choice for requirement [1] using jQuery's API's which are already tested
- Good choice for requirement [2] as long as the Coffeescript code is easy to follow
- Excellent choice for requirement [3] since the templating will keep this in mind

Cons:

- A little bit more complex
- More JavaScript

I had no choice but to go for Option 2, which also seems to be the more fun alternative by a landslide.

Justification of the used tools/gems

Excluding the gems that comes with "rails new app", in order of appearance:

haml_coffee_assets	For interpreting hamlc files with JST
exec_js	For generating html from templates with JST
haml-rails	For generating html from haml files
rspec-rails	Framework for rails unit testing
jasmine-rails	Framework for javascript unit testing
jasminerice	To run jasmine specs in the browser

factory_girl_rails I love FactoryGirl, but in retrospective adding it at this stage of the app was a huge overkill. But now it is there. No regrets.

timecop Also a personal favorite: Better testing with respect to time.

bootstrap-sass I'm not a CSS expert or a front page "artiste", I just wanted to get a quick css theme going.

rails_layout See above, or https://github.com/RailsApps/rails_layout

Project structure and why you chose it

I saw no reason to deviate from the project structure supplied by Rails. In fact, trying to fight is a path that only leads to Pain. I probably did not understand this question properly, and I should probably stop trying to answer it here, and instead offer to discuss this in person.

ISSUES - IMPORTANT!

I had to change the Sinatra app in order for the application to work. I encountered two problems:

1) Cross site http requests are blocked by the Sinatra app, which means I am not allowed to do a GET request to the Sinatra app from my rails app:
```
W, [2014-01-15T21:28:24.735104 #7991]  WARN -- : attack prevented by
Rack::Protection::JsonCsrf
```

2) The image paths for games in the Sinatra app was not correct and lead to broken images

So these were the two issues I ran in to and for both of them I decided that the best solution is to modify the server code, which should be OK since the CMS is simply another application that may also have bugs/issues (assumption [1]).

Solution:

1) Put `headers("Access-Control-Allow-Origin" => "*" )` in before do. Now, this is a security concern and I will flag it as such before merging to master so someone well versed in Sinatra can let me know how this actually should be solved.

2) Add "SERVER +" to correct the paths

PS: I put my modified version of server.rb in `vendor/server_copy.rb` for your reference.

Sincerely,

Mathias Klippinge