[**Chương 1: Từ số 0 đến deploy sản phẩm** 2](#_Toc433705500)

[**Chương 2: Ứng dụng Toy** 3](#_Toc433705501)

[**Chương 3: Xây dựng 1 trang web tĩnh** 3](#_Toc433705502)

[**Chương 4 : Rails - hương vị của Ruby** 10](#_Toc433705503)

[**Chương 5: Bố trí trang web** 11](#_Toc433705504)

[**Chương 6: Mô hình hóa Users** 11](#_Toc433705505)

[**Chương 7: Sign Up** 11](#_Toc433705506)

[**Chương 8: Login, logout** 11](#_Toc433705507)

[**Chương 9: Updating , Showing, and deleting users** 11](#_Toc433705508)

[**Chương 10: Account activation and password reset** 11](#_Toc433705509)

[**Chương 11: User micropost** 11](#_Toc433705510)

[**Chương 12: Following users** 11](#_Toc433705511)

Các bước tạo Rails App theo hướng dẫn tại trang: Ruby on Rails Tutorial tái bản lần 3

# **Chương 1: Từ số 0 đến deploy sản phẩm**

Tham khảo: <https://www.railstutorial.org/book/beginning>

Xin chào bạn đến với cuốn sách: “Ruby on Rails Tutorial:Lear Web Development with Rails” (Học làm web development với Rails).Mục đích của cuốn sách này là giúp bạn làm sao để tạo ra 1 web application tùy chỉnh bằng công cụ rails framework.

Box1.1 : Hạ thấp rào cản

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Ấn bản thứ ba của Ruby on Rails Tutorial nhằm mục đích giảm các rào cản để bắt đầu với Rails trong một số cách:

+ Sử dụng một môi trường phát triển tiêu chuẩn trong các đám mây (mục 1.2), mà sidesteps nhiều vấn đề liên quan đến cài đặt và cấu hình một hệ thống mới

+ Sử dụng Rails 'mặc định ngăn xếp', bao gồm cả việc xây dựng trong khuôn khổ kiểm tra minitest

+ Elimination của nhiều phụ thuộc bên ngoài (RSpec, dưa chuột, Capybara, Factory Girl)A trọng lượng nhẹ hơn và cách tiếp cận linh hoạt hơn để thử nghiệm Deferral hoặc loại bỏ các tùy chọn phức tạp hơn cấu hình (Spork, RubyTest)

+ Less nhấn mạnh vào các tính năng cụ thể cho bất kỳ phiên bản nào của Rails, với sự nhấn mạnh về nguyên tắc chung về phát triển web

Tôi hy vọng rằng những thay đổi này sẽ làm cho các ấn bản thứ ba của Ruby on Rails Tutorial tiếp cận với một đối tượng rộng hơn so với phiên bản trước đó.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Trong chương đầu tiên này, chúng ta sẽ bắt đầu với Ruby on Rails bằngcách cài đặt tất cả các phần mềm cần thiết và bằng cách thiết lập môi trường phát triển của chúng tôi (Section - Mục 1.2). Sau đó chúng tôi sẽ tạo ra ứng dụng Rails đầu tiên của chúng tôi, được gọi là “hello\_app”. The Rails Tutorial nhấn mạnh thực hành phát triển phần mềm tốt, vì vậy ngay sau khi tạo mới dự án Rails mới của chúng tôi, chúng tôi sẽ đặt nó dưới sự kiểm soát các phiên bản với Git (Mục 1.4). Và, tin có thể khó tin nhưng trong chương này chúng tôi thậm chí sẽ đưa ứng dụng đầu tiên của chúng tôi trên trang web rộng hơn bằng cách triển khai(deploying) nó vào sản xuất (Mục 1.5)

Trong chương 2, chúng tôi sẽ thực hiện một dự án thứ hai, mà mục đích là để minh họa các hoạt động cơ bản của một ứng dụng Rails. Để có được và chạy một cách nhanh chóng, chúng tôi sẽ xây dựng ứng dụng này “toy\_app”(ứng dụng đồ chơi, toy trong tiếng anh là đồ chơi) sử dụng scaffold(scaffold trong tiếng anh nghĩa là dàn , khan đài , dựng dàn, làm dàn – xây dựng khung có sẵn) (Box - Hộp 1.2) để tạo ra mã. Vì mã này vừa xấu xí và phức tạp nên Chương 2 sẽ tập trung vào việc tương tác với app “toy\_app” thông qua URI của nó (thường được gọi là URL)bằng cách sử dụng một trình duyệt web.

Phần còn lại của hướng dẫn tập trung vào việc phát triển một ứng dụng mẫu sản lớn duy nhất gọi là “sample\_app”, viết tất cả các mã từ đầu. Chúng tôi sẽ phát triển các ứng dụng mẫu sử dụng một sự kết hợp của mockups(giả lập), kiểm tra phát triển bắt buộc (Test Divent Development - TDD), và kiểm tra liên tục (integration test). Chúng tôi sẽ bắt đầu trong Chương 3 bằng cách tạo ra các trang web tĩnh và sau đó thêm một chút nội dung động. Chúng tôi sẽ đi đường vòng nhanh trong Chương 4 để tìm hiểu một chút về ngôn ngữ Ruby – ngôn ngữ phát triển của Rails. Sau đó, trong chương 5 đến Chương 10, chúng tôi sẽ hoàn thành nền tảng cho các ứng dụng sample\_app bằng cách làm site layout, data model , và chức năng đăng ký và xác thực người dùng hệ thống (bao gồm cả tài khoản và kích hoạt reset mật khẩu). Cuối cùng, trong Chương 11 và Chương 12, chúng tôi sẽ thêm microblogging và các tính năng xã hội để tạo công việc cho site ví dụ này.

BOX 1.2. Scaffold : Nhanh , đơn giản và quyến rũ

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Ngay từ đầu, Rails đã được hưởng lợi từ một cảm giác sờ thấy hứng thú, bắt đầu với sự nổi tiếng 15 phút weblog video bằng Rails tác giả David Heinemeier Hansson. Đó video và kế thừa của nó là một cách tuyệt vời để có được một hương vị của điện Rails, và tôi khuyên bạn nên xem chúng. Nhưng được cảnh báo: họ thực hiện điều tuyệt vời mười lăm phút chiến công của họ bằng cách sử dụng một tính năng gọi là scaffold , mà chủ yếu dựa vào mã được tạo ra, kỳ diệu được tạo ra bởi các Rails tạo lệnh scaffold .

Khi viết một Ruby on Rails hướng dẫn, nó là hấp dẫn dựa trên các scaffold cách tiếp cận nó nhanh hơn, dễ dàng hơn, quyến rũ hơn. Nhưng sự phức tạp và số lượng tuyệt đối của các mã trong các scaffold có thể hoàn toàn áp đảo để một nhà phát triển đầu Rails; bạn có thể sử dụng nó, nhưng có thể bạn sẽ không hiểu nó. Theo cách tiếp cận giàn giáo có nguy cơ biến bạn thành một máy phát điện script bậc thầy với chút (và giòn) kiến ​​thức thực tế của Rails.

Trong Ruby on Rails Tutorial, chúng tôi sẽ lấy (gần như) cách tiếp cận đối cực: mặc dù Chương 2 sẽ phát triển một ứng dụng đồ chơi nhỏ, sử dụng giàn giáo, cốt lõi của Rails Tutorial là ứng dụng mẫu, mà chúng ta sẽ bắt đầu viết trong Chương 3. Ở mỗi giai đoạn phát triển các ứng dụng mẫu, chúng tôi sẽ viết nhỏ, cắn cỡ mẩu mã đơn giản, đủ để hiểu, nhưng vẫn đủ để cuốn tiểu thuyết được thử thách. Hiệu quả tích lũy sẽ là một kiến ​​thức linh hoạt hơn sâu sắc hơn về Rails, tạo cho bạn một nền tảng tốt cho văn bản gần như bất kỳ loại ứng dụng web.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Như vậy ta sẽ bỏ qua chương 1,2 và vào luôn chương 3. Vì từ chương 3 trở đi sẽ nói rõ hơn về cách phát triển từng bước với 1 rails app chứ không sử dụng scaffold(giàn giáo) để sinh code tự động.

* 1. **Giới thiệu**
     1. **Điều kiện tiên quyết**
     2. **Quy ước trong cuốn sách này**
  2. **Up và chạy ứng dụng**
     1. **Môi trường phát triển**
     2. **Cài đặt Rails**
  3. **Ứng dụng đầu tiên**
     1. **Bundler**
     2. **Rails server**
     3. **MVC**
     4. **HelloWorld**
  4. **Quản lý Version với Git**
     1. **Cài đặt và cấu hình**
     2. **Những điều tốt mà Git làm cho bạn?**
     3. **Bitbucket**
     4. **Branch, edit , commit, ,merge**
  5. **Deploying**
     1. **Cài đặt Heroku**
     2. **Phát triển Heroku, bước đầu tiên**
     3. **Phát triển Heroku, bước thứ hai**
     4. **Heroku commands**
  6. **Kết luận**
     1. **Chúng ta học được gì ở chương này?**
  7. **Exercies**

# **Chương 2: Ứng dụng Toy**

Tham khảo: <https://www.railstutorial.org/book/toy_app>

# **Chương 3: Xây dựng 1 trang web tĩnh**

Trong chương này, chúng ta sẽ bắt đầu phát triển các ứng dụng mẫu chuyên nghiệp cấp mà sẽ phục vụ như là ví dụ của chúng tôi trong suốt phần còn lại của hướng dẫn này. Mặc dù các ứng dụng mẫu cuối cùng sẽ có người dùng, microposts, và đăng nhập đầy đủ và khung xác thực, chúng ta sẽ bắt đầu với một chủ đề dường như hạn chế: việc tạo ra các trang tĩnh. Mặc dù đơn giản rõ ràng của nó, làm cho các trang tĩnh là một bài tập rất cao tính hướng dẫn, giàu ý nghĩa, một sự khởi đầu hoàn hảo cho các ứng dụng non trẻ của chúng tôi.

Mặc dù Rails được thiết kế để làm cho các trang web động cơ sở dữ liệu hậu thuẫn, nó cũng xuất sắc trong việc làm các loại của các trang tĩnh, chúng tôi có thể làm với các tập tin HTML thô. Trong thực tế, sử dụng Rails cho cả các trang tĩnh mang lại một lợi thế khác biệt: chúng ta có thể dễ dàng thêm chỉ một lượng nhỏ nội dung động. Trong chương này chúng ta sẽ tìm hiểu làm thế nào. Trên đường đi, chúng ta sẽ nhận được hương vị đầu tiên của chúng tôi kiểm tra tự động, mà sẽ giúp chúng tôi tự tin hơn

rằng mã của chúng tôi là chính xác. Hơn nữa, có một bộ kiểm tra tốt sẽ cho phép chúng ta cấu trúc lại mã của chúng tôi với sự tự tin, thay đổi hình thức của nó mà không thay đổi chức năng của nó.

>rails new sample\_app

Như trong 2 chương trước thì ta đã chạy lệnh >bundle install để thực thi cài đặt các gem được khai báo trong Gemfile. Vì vậy để bỏ qua việc cài đặt gem production ta sẽ dùng lệnh

>bundle install --without production

Việc bỏ qua như vậy cũng đồng thời giúp ta bỏ qua gem pg được sử dụng với PostgreSQL. Vì vậy ta sẽ sử dụng Sqlite như mặc định. Mặc dù trong môi trường sản xuất(product) thì việc sử dụng sqlite là không được khuyến khích nhưng để dễ dàng tiếp cận thì ta cũng chưa cần quan tâm tới điều này, tránh những phức tạp trong cài đặt

Ta có thể sử dụng lệnh update trực tiếp thay vì thông qua Gemfile

>bundle update

<Đang xem tới 3.2 Static Pages>

https://www.railstutorial.org/book/static\_pages

**3.1: Sample setup**

Tạo 1 app tên là sample\_app

>rails \_rails\_version\_ new app\_name

vd: rails \_4.2.2\_ new sample\_app

Phần sau có hướng dẫn việc deploy app lên git nhưng mình tạm thời ko quan tâm đến vấn đề này.

**3.2: Generate static page**

Tạo 1 controller có tên là StaticPagesController, ta sử dụng

>rails generate controller StaticPages home help

Ngoài ra, Rails còn hỗ trợ 1 số cách viết tắt như là:

$>rails generate = $>rails g

$>rails server = $>rails s

$>rails console = $>rails c

$>bundle install = $>bundle

$>rake test = $>rake

Bạn nên nhớ để sử dụng cho thuận tiện, hoặc sẽ hiểu khi nó được sử dụng trong 1 số tài liệu

Muốn undo lại quá trình làm thì bạn có thể destroy controller tạo ra, nhờ sự hỗ trợ này của Rails mà ta có thể dễ dàng xóa đi các file tự động sinh ra mà ko sợ bỏ xót như khi xóa bằng tay

$>rails destroy controller StaticPages home help

Khi sử dụng generate thì rails cũng tự động tạo ra các routing bằng cách khai báo vào trong file config router như sau:

get 'static\_pages/home' # Thực thi 1 đường dẫn từ controller tới action home

get 'static\_pages/help'

Tiếp theo ta thay đổi nội dung của view như trong hướng dẫn

**3.3: Bắt đầu với testing**

Sau khi tạo ra và thay đổi nội dung cho các trang Home và Help cho app sample\_app, bây giờ chúng ta sẽ thêm một trang About . Khi thực hiện một sự thay đổi tự nhiên như vậy, nó là một thực hành tốt để viết một bài kiểm tra tự động để xác minh rằng các tính năng này được thực hiện một cách chính xác. trong quá trình xây dựng và Phát triển một ứng dụng, các bộ kiểm tra kết quả (test suite) phục vụ như là một mạng lưới an toàn và là tài liệu hướng dẫn thực thi của mã nguồn ứng dụng. Khi thực hiện quyền, viết bài kiểm tra cũng cho phép chúng ta phát triển nhanh hơn bất chấp yêu cầu mã thêm, bởi vì chúng tôi sẽ kết thúc lãng phí thời gian ít cố gắng theo dõi lỗi.

Điều này đúng một lần duy nhất chúng tôi có được tốt ở các bài kiểm tra văn bản, tuy nhiên, đó là một lý do quan trọng là phải bắt đầu thực hành càng sớm càng tốt.

Mặc dù hầu như tất cả các nhà phát triển đồng ý rằng thử nghiệm Rails là một ý tưởng tốt, có một sự đa dạng của các quan điểm về các chi tiết. Có một cuộc tranh luận sôi nổi về việc đặc biệt là việc sử dụng các "thử nghiệm điều khiển phát triển" (Tes-driven development :TDD),một kỹ thuật xét nghiệm trong đó các lập trình viên viết các bài kiểm tra thất bại đầu tiên, và sau đó viết mã ứng dụng để có được các xét nghiệm để vượt qua.

Ruby on Rails Tutorial mất một cách tiếp cận trực quan nhẹ để thử nghiệm, sử dụng TDD khi thuận tiện mà không giáo điều về nó.

Box3-3: Khi nào thì thực hiện test (đưa ra lợi ích của Test, khi nào thì cần test)

**3.3.1. Test đầu tiên**

Vấn đề test được nói trên trong box 3-3 có vẻ thật sự phức tạp với những ng ko có kiến thức sâu và rộng

về cả Rails lẫn Ruby, tuy nhiên thật may mắn là "rails generate controller" đã cung cấp cho chúng ta

các file test nằm trong thư mục test

Kiểm tra file test có tồn tại

>ls test/controllers/

#=> static\_pages\_controller.rb

Mở file test và xem nội dung của nó:

===================================

test "should get home" do

get :home

assert\_response :success #assert : xác nhận , response: phản hồi

end

===================================

Vào thời điểm lúc này ko quan trọng để hiểu cú pháp được viết trong file test nhưng ta có thể dễ dàng

thấy có 2 phần kiểm tra và mỗi bài kiểm tra có khai báo như trên. Mỗi bài kiểm tra bao gồm hành động

và thẩm tra(thông qua 1 khẳng định là kết quả có thành công). Việc sử dụng get chỉ ra rằng việc test

của chúng tôi mong đợi việc truy cập đến Home và Help để đc các trang web thông thường , và nó mong đợi

1 xác nhận thành công.

Và ta thực hiện test

>bundle exec rake test

Ta thử thêm 1 action about vào và thực hiện test theo các mục 3.3.2 , 3.3.3 và 3.3.4

**3.4 : Thêm một chút động cho trang web**

Như các phần trước thì ta đã tạo ra action và view cho trang web. Vậy bây giờ ta sẽ làm cho nó động 1 chút(slightly dynamic) bằng cách thay đổi title tương ứng với từng page. Đây sẽ là tiền đề để chúng ta học ở chương 7 về dynamic content

Kế hoạch đặt ra trong chương này đó là chúng tôi sẽ chỉnh sửa các trang home, help và about để title của chúng sẽ tương ứng với mỗi trang. Điều này liên quan đến việc sử dụng các thẻ <title>. Hầu hết các trình duyệt hiển thị nội dung của title ở trên khung cửa sổ của trình duyệt và nó cũng rất quan trọng để tối ưu hóa công cụ tìm kiếm. Chúng tôi sẽ sử dụng "Red , Green và Refactor" theo vòng đời chu kì sau:

+ Red : Ta sẽ thêm các simple tests cho 1 trang của chúng tôi

+ Green : Tôi sẽ đi thêm title cho 3 trang Home, Help và About

+ Refactor: Loại bỏ sự trùng lặp giữa các trang, và cuối cùng tất cả các title sẽ có dạng

"<page\_title> | Ruby on Rails Tutorial Sample App "

Khi ta create rails app thì đồng thời luôn có 1 layout defaut , nhưng tót hơn hết là ta nên đổi tên nó vì nó được sử dụng để quản lý các thiết lập cơ bản của layout trong rails, nên để tránh việc nó làm ảnh hưởng thì tạm thời ta sẽ đổi tên nó.

Lệnh mv trong Linux : mv [Option] [Nguồn] [Đích]

(mv là 1 lệnh đơn giản trong linux dùng để di chuyển file thư mục từ vị trí cũ đến vị trí mới.)

Đôi khi mv còn được sử dụng để đổi tên file , tên thư mục

>mv app/views/layouts/application.html.erb layout\_file

Như vậy file layout\_file được đổi tên sẽ nằm ở thư mục gốc của app

Để kiểm tra sự hiện diện của 1 thẻ html nào đó, ta sử dụng assert\_select

Xem bảng 3-22: Kiểm tra title của các page Home ,Help và About

Thực hiện test

**3.4.3 : Nhúng mã Ruby vào code HTML**

Chúng ta đã có thể tạo ra 1 web tĩnh với controller và action, nhưng html thì hoàn toàn tĩnh nên nó ko thể thể hiện được sức mạnh của Rails. Hơn nữa nó bị lặp lại 1 cách khủng khiếp

Ta có thể nhận thấy những điều sau đây:

+ Title được lặp lại gần như toàn bộ "Ruby on Rails Tutorial Sample App" được áp dụng cho cả 3 title

+ Toàn bộ cấu trúc HTML được lặp lại trên mỗi trang

Mà việc lặp lại mã đã vi phạm 1 trong những nguyên tắc quan trong của Rails đó là DRY (Không lặp lại). Như vậy trong phần này, ta sẽ đi làm 1 việc cần thiết đó là "Loại bỏ sự lặp lại" bằng cách sử dụng việc nhúng mã Ruby vào trong HTML mà Rails đã hỗ trợ.Ta chú ý tới bảng 3-28 sẽ thực hiện việc nhúng mã Ruby vào html và thay thế chũ "Home" trong title .

Tiếp theo ta sẽ lần lượt thiết lập title cho các page như theo hướng dẫn

Cấu trúc chung cho các trang :

========================================================

<% provide(:title, "The Title") %>

<!DOCTYPE html>

<html>

<head>

<title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>

</head>

<body>

Contents

</body>

</html>

========================================================

Tuy nhiên để cấu trúc phổ biến này được thiết lập , Rails cần sử dụng đến 1 tập tin được bố trí đặc biệt

đó là application.html.erb. Lúc trước ta đã đổi tên của file này, giờ ta sẽ khôi phục lại nó

> mv layout\_file app/views/layouts/application.html.erb

Ta cũng sẽ thay thế title của file này như với các ví dụ trên (kết quả như bảng 3-32)

Chú ý tới 1 dòng đặc biệt : <%= yield %>

Mã này chịu trách nhiệm cho việc chèn thêm các nội dung của page vào trong layout.Ta ko quan trọng việc biết chính xác cách nó làm việc nhưng điều quan trọng là sử dụng cách bố trí này đảm bảo rằng, ví du muốn truy cập vào trang static\_pages/home thì nó sẽ đảm bảo chuyển đổi nội dung của home.html.erb sang mã html và chèn thêm vào chuỗi <%= yield %>

Tiếp theo ta cần chú ý tới 1 số mã được khai báo :

<%= stylesheet\_link\_tag ... %>

<%= javascript\_include\_tag "application", ... %>

<%= csrf\_meta\_tags %>

Nó được khai báo để có thể sử dụng các file css, javascript trong thư mục asset. Còn <%= csrf\_meta\_tags %>

được sử dụng để ngăn chặn cross-site từ các web độc hại giả mạo yêu cầu .

Ta sẽ loại bỏ tất cả các mã html ko cần thiết như trong 3-33,3-34 và 3-35

Thực hiện test xem có vấn đề gì xảy ra hay ko

**3.4.4. thực hiện thiết lập root route**

Sửa trong file

root 'static\_pages#home' #=> thiết lập home thành trang chủ

# **Chương 4 : Rails - hương vị của Ruby**

Như căn cứ vào chương 3, chương 4 sẽ đi tìm hiểu 1 số yếu tổ của Ruby là quan trong với Rails

**4.1. Động lưc**

Nếu không hiểu về Ruby , thì gần như bạn chẳng hiểu từ chương 1 tới chương 3 Rails đã thực hiện những gì để có thể hoạt động.

Sử dụng CSS

stylesheet\_link\_tag 'application', media: 'all', 'data-turbolinks-track' => true

tương đương với

<link data-turbolinks-track="true" href="/assets/application.css" media="all"

rel="stylesheet" />

# **Chương 5: Bố trí trang web**

# **Chương 6: Mô hình hóa Users**

# **Chương 7: Sign Up**

# **Chương 8: Login, logout**

# **Chương 9: Updating , Showing, and deleting users**

# **Chương 10: Account activation and password reset**

# **Chương 11: User micropost**

# **Chương 12: Following users**