Intro to Ruby on Rails

Really? How many slides? -- 72

Ruby on Rails

- bundler
- scaffolding, and generators
- polyglot; haml and sass
- Yaml, JSON, not XML
- Testing, Rspec, Cucumber

Ruby on Rails

Was created by David Heinemeier Hansson as a kind of byproduct of Basecamp’s development at 37signals in 2004.

Basecamp was built in Ruby because Hansson found PHP and Java not powerful or flexible enough.

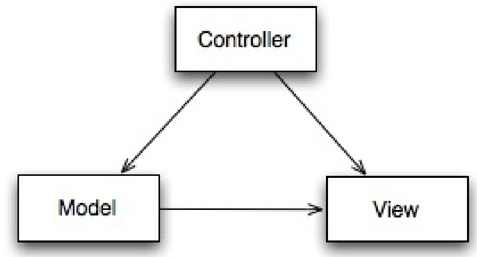
Rails is founded on pragmatism and established paradigms instead of exotic new ideas. And that’s what made it so successful.

Rails is based on the Model-View-Controller pattern.

The Models are your business objects describing the structure and behavior of the problem your application is trying to solve. These are usually backed by an Object-Relational-Mapping framework that persists your objects to a database in the background.

The Views are the templates that render data to the user and all the logic surrounding presentational aspects of your app.

- MVC Framework



The Controller is repsonsible for the control flow of the application.

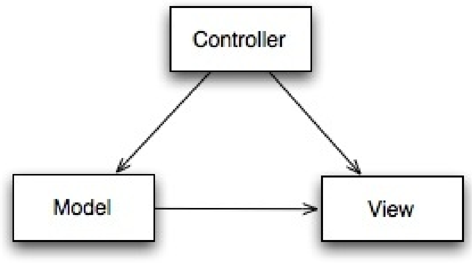
The Model contains business data and behaviour.

The View renders Models to the user, triggered by the controller.

Model-View-Controller (cont)

The Controller sits at the heart of everything, processing requests from clients, initiating changes in the models and triggering the rendering of the templates.

- MVC Framework



The Controller is repsonsible for the control flow of the application.

The Model contains business data and behaviour.

The View renders Models to the user, triggered by the controller.

Rails is “opinionated software.”

It doesn’t want to be everything for everyone. It focuses on one way of doing things and streamlines all its parts around that way.

That’s not to say there’s no possibility of doing things differently if you need to, but you’ll definitely have it easier if you do things “the Rails way.”

Programmer productivity was the main goal during Rails’ development, not performance.

- Convention over Configuration
- KIS
- Testable

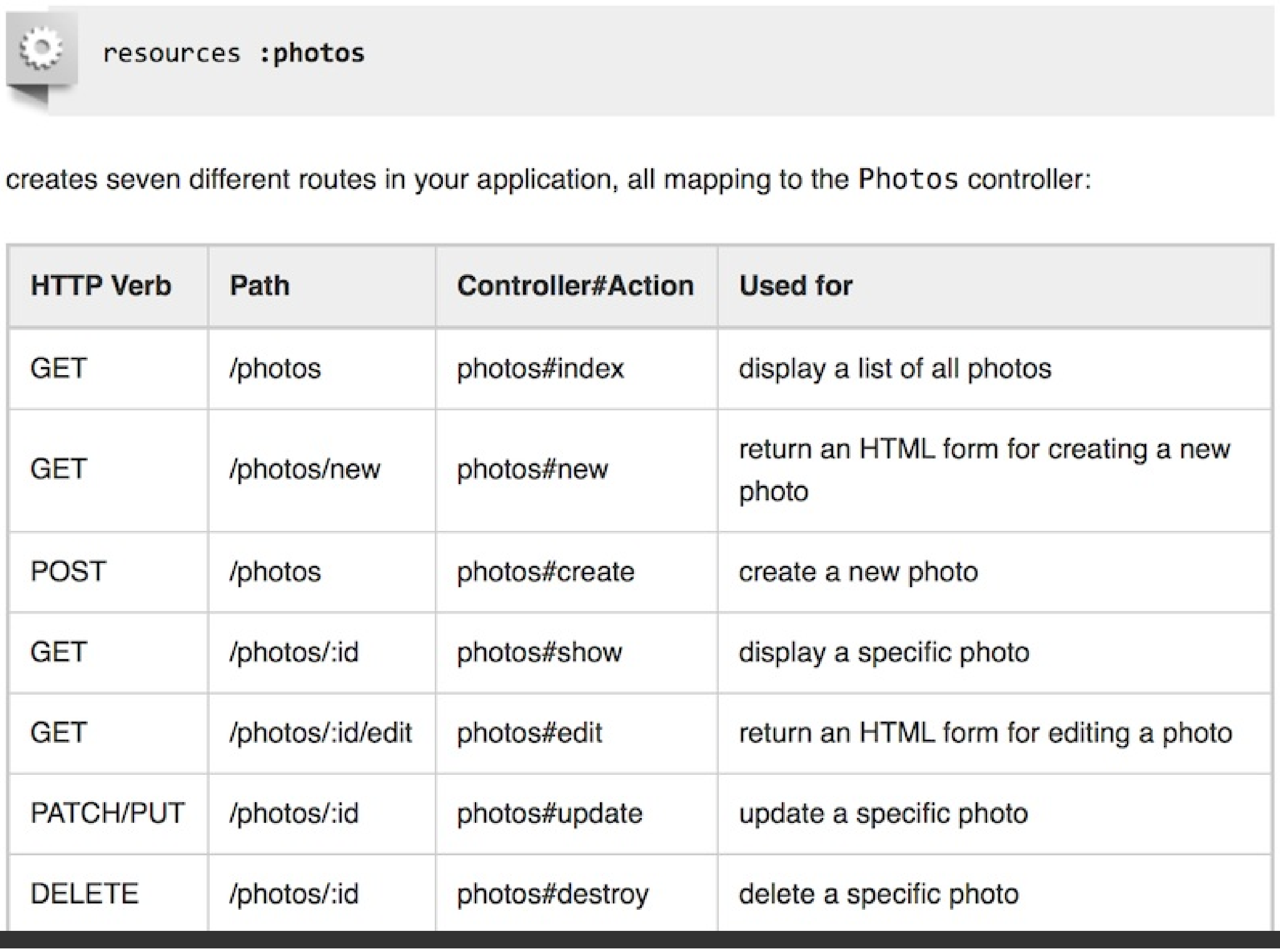
Readability Clarity, KIS!

Polyglot Ruby ; CoffeeScript Haml and Sass

multiple programming languages and multiple "modularity paradigms" in application development. (!GWT !node.js and javascript everywhere)

- CoffeeScript
- Js2coffee
- Haml
- Sass

REST



```
resources :photos
```

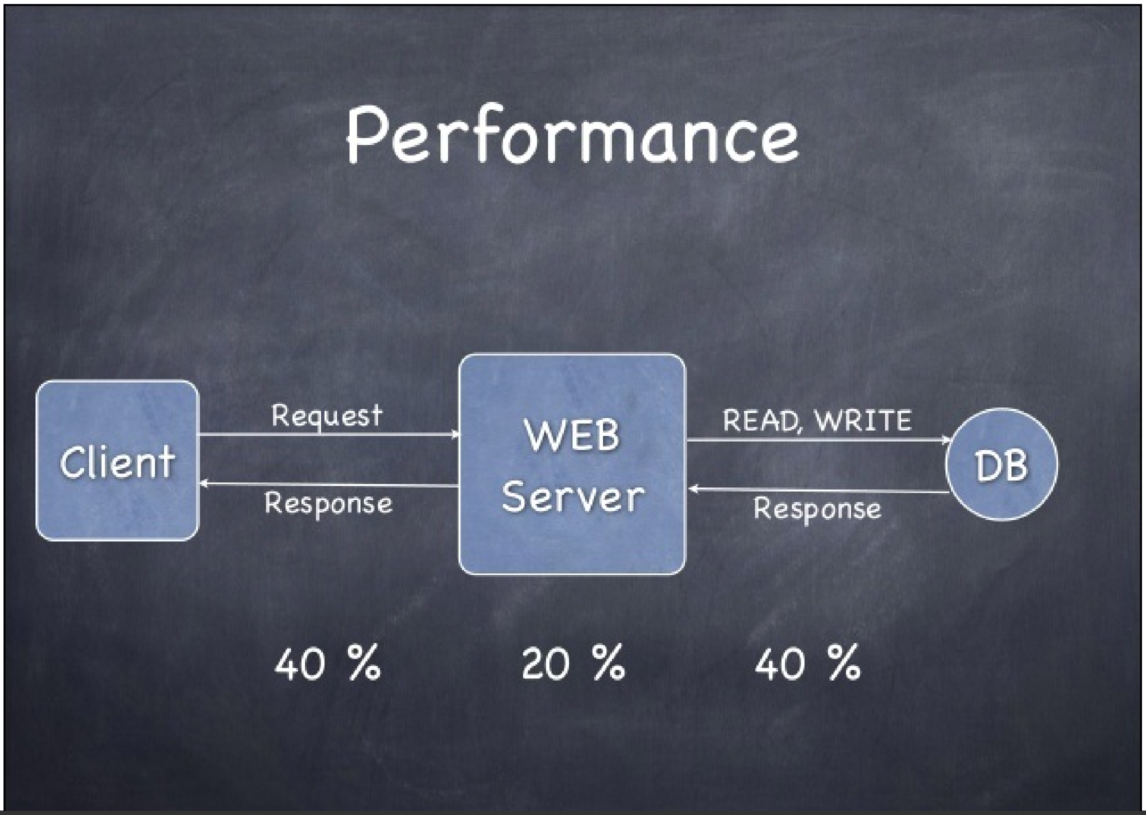
creates seven different routes in your application, all mapping to the `Photos` controller:

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

Performance

Ruby is slower than Java!

True! But ...



Get to know the tools

Text Editor

Textmate 2, Sublime Text, Vim and Emacs are examples of text editors your can use for writing code and editing files.

Terminal

Bash, Zsh, Where you start the rails server and run commands.

Web browser

(Firefox, Safari, Chrome) for viewing your application.

Rails Tutorial

Tutorial

based on Rails Girls Guides

Creating the application

We’re going to create a new Rails app called Idea. using Bootstrap-sass

Install Rails

```
➜  gem install rails

➜  rails -v
Rails 4.1.6

➜  sqlite3 --version
3.7.13 2012-07-17 17:46:21 ...
```

Next, create a new Application

```
mkdir projects

cd projects

rails new idea -T

cd idea
```

Rails app dir structure.

File/	Folder Purpose
app/	Contains the controllers, models, views, helpers, mailers and assets for your app.
bin/	Contains scripts that starts your app
config/	Configure your application's routes, database, and more.
config.ru	Rack configuration for Rack based servers used to start the application.
db/	Contains your current database schema, as well as the database migrations.

Gemfile Gemfile.lock	These files allow you to specify what gem dependencies are needed for your Rails application.

Rails app dir structure.

```
lib/          Extended modules for your application.
log/          Application log files.
public/       The only folder seen by the world as-is.
              Contains static files and compiled assets.
Rakefile      This file locates and loads tasks that
              can be run from the command line.

README.rdoc This is a brief instruction manual for
              your application. You should edit this
              file to tell others what your application
              does, how to set it up, and so on.
test/         Unit tests, fixtures, and other test files.
tmp/          Temporary files (cache, pid, session files)
vendor/       A place for all third-party code.
              In a typical Rails application this
              includes vendored gems.
```

Rake

```
rake routes

rake -T
```

Bundler, RubyGems

Edit Gemfile, after:

```
gem 'sass-rails', '~> 4.0.3'
```

add:

```
gem 'haml-rails',              '~> 0.5'
gem 'bootstrap-sass',          '~> 3.2'
gem 'autoprefixer-rails',      '~> 3.1'
```

Near end of file add:

```
group :development do
  gem 'awesome_print',       '~> 1.2.0'
  gem 'pry-rails',           '~> 0.3.2'
  gem 'pry-nav',             '~> 0.2.3'
  gem 'better_errors',       '= 1.1.0'
  gem 'binding_of_caller',   '= 0.7.2'
end
```

Install Gems for rails app.

```
bundle install
```

Add to Git

```
git init

git add .

git commit -m "Initial Rails App Generation"

rails s
```

Edit `config/application.rb` after the line

```
module Collectable
  class Application < Rails::Application
```

add;

```
  config.sass.preferred_syntax = :sass
```

Open http://localhost:3000 in your browser.

You should see “Welcome aboard” page, which means that the generation of your new app worked correctly.

Hit **CTRL-C** in the terminal to quit the server.

Create Idea scaffold

We're going to use Rails' scaffold functionality to generate a starting point that allows us to list, add, remove, edit, and view things; in our case ideas.

```
rails generate scaffold idea name:string \
    description:text picture:string
```

The scaffold creates new files in your project directory, but to get it to work properly we need to run a couple of other commands to update our database and restart the server.

```
bin/rake db:migrate
rails server
```

Open http://localhost:3000/ideas in your browser.

```
git add .
git commit -m "scaffolded ideas"
```

Environments

```
export RAILS_ENV=development

export RAILS_ENV=test

export RAILS_ENV=production
```

Database configuration; `config/database.yml`

```
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

test:
  <<: *default
  database: db/test.sqlite3

production:
  <<: *default
  database: db/production.sqlite3
```

Design

What part of views is HTML and what is Embedded Ruby (ERB)?

What is MVC and how does this relate to it?

(Models and controllers are responsible for generating the HTML views.)

The app doesn’t look very nice yet. Let’s do something about that.

We’ll use the Twitter Bootstrap project to give us nicer styling really easily.

Import Bootstrap styles in `app/assets/stylesheets/application.css.scss`

```
@import "bootstrap-sprockets";
@import "bootstrap";
```

Require Bootstrap Javascripts in `app/assets/javascripts/application.js`

```
//= require jquery
//= require bootstrap-sprockets
```

Open `app/views/layouts/application.html.erb`

```
<!DOCTYPE html>
<html>
<head>
  <title>Ideas</title>
  <%= stylesheet_link_tag    'application',
    media: 'all', 'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application',
    'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
</head>
<body>

<%= yield %>

</body>
</html>
```

Rename it to `app/views/layouts/application.html.haml`

```
git mv app/views/layouts/application.html.erb
   app/views/layouts/application.html.haml
```

```
!!!
%html
  %head
    %title Ideas
    = stylesheet_link_tag    'application',
        media: 'all', 'data-turbolinks-track' => true
    = javascript_include_tag 'application',
        'data-turbolinks-track' => true
    = csrf_meta_tags
  %body
    = yield
```

and replace

```
= yield
```

with

```
.container
  = yield
```

Let’s also add a navigation bar and footer to the layout.

In the same file, under `%body` add

```
%nav.navbar.navbar-default.navbar-fixed-top{
    :role => "navigation"}
  .container
    .navbar-header
      %button.navbar-toggle{
          "data-target" => ".navbar-collapse",
          "data-toggle" =>  "collapse",
          :type => "button"}
        %span.sr-only Toggle navigation
        %span.icon-bar
        %span.icon-bar
        %span.icon-bar
      %a.navbar-brand{:href => "/"} The Idea app
    .collapse.navbar-collapse
      %ul.nav.navbar-nav
        %li.active
          %a{:href => "/ideas"} Ideas
```

and after `%body` add

```
%footer
  .container
    Hickory Ground 2014
%script{:src => "//railsgirls.com/assets/bootstrap.js"}
```

Now let's also change the styling of the ideas table. Open `app/assets/stylesheets/application.scss` and at the bottom add

```
body { padding-top: 100px; }
footer { margin-top: 100px; }
table, td, th { vertical-align: middle; border: none; }
th { border-bottom: 1px solid #DDD; }
```

Now make sure you saved your files and refresh the browser to see what was changed. You can also change the HTML & CSS further.

In case your Terminal shows you an error message that sort of implies there is something wrong with your JavaScript or CoffeeScript, install nodejs. This issue should not appear when you've used the RailsInstaller (but when you've installed Rails via gem install rails).

Adding picture uploads

We need to install a piece of software to let us upload files in Rails.

Open Gemfile in the project directory using your text editor and under the line

```
gem 'sqlite3'
```

add

```
gem 'carrierwave'
```

In the terminal run:

```
bundle
```

Now we can generate the code for handling uploads. In the terminal run:

```
rails generate uploader Picture
```

At this point you need to restart the Rails server process in the terminal.

Hit CTRL-C in the terminal to quit the server. Once it has stopped, you can press the up arrow to get to the last command entered, then hit enter to start the server again.

This is needed for the app to load the added library.

Open app/models/idea.rb and under the line

```
class Idea < ActiveRecord::Base
```

add

```
mount_uploader :picture, PictureUploader
```

Open app/views/ideas/_form.html.haml and change

```
= f.text_field :picture
```

to

```
= f.file_field :picture
```

Sometimes, you might get an `TypeError: can’t cast ActionDispatch::Http::UploadedFile to string.`

If this happens, in file `app/views/ideas/_form.html.haml` change the line

```
= form_for(@idea) do |f|
```

to

```
= form_for @idea, :html => {:multipart => true} do |f|
```

In your browser, add new idea with a picture. When you upload a picture it doesn't look nice because it only shows a path to the file, so let's fix that.

Open `app/views/ideas/show.html.haml` and change

```
= @idea.picture
```

to

```
= image_tag(@idea.picture_url, :width => 600) if @idea.
```

Now refresh your browser to see what changed.

Fine tune the routes

Open http://localhost:3000. It still shows the “Welcome aboard” page.

Let’s make it redirect to the ideas page.

Open `config/routes.rb` and after the first line add

```
root :to => redirect('/ideas')
```

Test the change by opening the root path (that is, http://localhost:3000/ or your preview url) in your browser.

Rails 3 users: You will need to delete the index.html from the /public/ folder for this to work.

Edit `.gitignore` and add dir `public/uploads` at end

save code

```
git add .
```

or

```
git add -p

git commit -m "Add picture uploader"
```

Create static page in your app

Lets add a static page to our app that will hold information about the author of this application — you!

```
rails generate controller pages info
```

This command will create you a new folder under app/views called /pages and under that a file called `info.html.haml` which will be your info page.

It also adds a new simple route to your routes.rb.

```
get "pages/info"
```

Now you can open the file `app/views/pages/info.html.haml` and add information about you in HTML.

```
git add .

git commit -m "Add info page"
```

Test your app with RSpec

RSpec is a Ruby testing framework, that describes our application’s behavior in a syntax that doesn’t look much like Ruby.

It outputs test results in your terminal.

Install Rspec

For starters, let’s install RSpec and all of its dependencies.

Edit `Gemspec`, add the line;

```
group :test, :development do
  gem 'rspec-rails',        '~> 3.0.0'
end
```

Then we call

```
bundle
rails generate rspec:install
```

in our project directory. This creates `rails_helper.rb`, `spec_helper.rb` in the `spec` folder, and `.rspec`.

Rubyists often use the words ‘test’ and ‘specification’ interchangeably, that’s why you’ll store your tests in the ‘specs’ folder. Save your test as `idea_spec.rb` (`<name_of_spec>_spec.rb`).

Inside that new file, write:

Next, let’s describe one of our specifications

```
describe Idea do
  it "has a name" # your examples (tests) go here
end
```

In your terminal run

```
rspec spec
```

which will output that your test is pending as it’s not yet implemented.

Let’s do something about that!

```
describe Idea do
  it "has a name" do
    idea = Idea.new
    expect(:name).to be
  end
end
```

should give you a more satisfying output.

Marking to-do's with tests

Yeah! To-do lists. Awesome. A nifty RSpec feature is the functionality to mark certain tests as pending.

Leaving out the `do` and the `end` in the example body, like so

```
it "has a title"
```

will mark a test as pending. For bigger applications, where you want to tackle one test at a time, you can also add an x in front of an example, making it read

```
describe Idea do
  xit "has a title" do
end
```

or use the word pending in your example.

run rspec with the flag; (can be added to .rspec)

```
--format documentation
```

Happy testing!

Resources

- Ruby on Rails
- Rails Tutorial
- Michael Hartl's Rails Tutorial
- RSpec Rails 3.1 docs
- Cucumber

Resources

- CoffeeScript
- Js2coffee
- Haml
- Sass
- Heroku
- Dot Cloud
- Travis CI

Exercises:

Commenting for Ideas

We are going to add the possibility to comment on ideas in your application.

The instructions for installing rails and building the ideas app can be found here.

Create comment scaffold

Create a comment scaffold, with the commentator name, the comment body (contents of the comment) and with the reference to the ideas table (`idea_id`).

```
rails g scaffold comment user_name:string body:text idea_id:integer
```

This will create a migration file that lets your database know about the new comments table. Run the migrations using

```
rake db:migrate
```

Add relations to models

You need to make sure that Rails knows the relation between objects (ideas and comments). As one idea can have many comments we need to make sure the idea model knows that. Open `app/models/idea.rb` and after the row

```
class Idea < ActiveRecord::Base
```

add

```
has_many :comments
```

The comment also has to know that it belongs to an idea. So open `app/models/comment.rb` and after

```
class Comment < ActiveRecord::Base
```

add the row

```
belongs_to :idea
```

Render the comment form and existing comments

Open `app/views/ideas/show.html` and after the `image_tag`

```
= image_tag(@idea.picture_url, :width => 600) if @idea.picture.prese
```

add

```
%h3 Comments
- @comments.each do |comment|
  %div
    %strong= comment.user_name
    %br
    %p= comment.body
    %p= link_to 'Delete', comment_path(comment), \
        method: :delete, data: { confirm: 'Are you sure?' }
= render 'comments/form'
```

In `app/controllers/ideas_controller.rb` add to show action after the row

```
@idea = Idea.find(params[:id])
```

this

```
@comments = @idea.comments.all
@comment = @idea.comments.build
```

Open `app/views/comments/_form.html.erb` and after

```
.field
  = f.label :body
  %br
  = f.text_area :body
```

add the row

```
= f.hidden_field :idea_id
```

next, remove

```
.field
  = f.label :idea_id
  %br
  = f.number_field :idea_id
```

That’s it. Now view an idea you have inserted to your application and there you should see the form for inserting a comment as well as deleting older comments.

Create thumbnails with Carrierwave

Installing ImageMagick

OS X: run `brew install imagemagick`. If you don’t have the brew command, you can install Homebrew here. Linux: On Ubuntu and Debian, run `sudo apt-get install imagemagick`. Use the appropriate package manager instead of apt-get for other distributions.

Open Gemfile in the project and add

```
gem 'mini_magick', '3.8.0'
```

under the line

```
gem 'carrierwave'
```

In the Terminal run:

```
bundle
```

Telling our app to create thumbnails when an image is uploaded

Open `app/uploaders/picture_uploader.rb` and find the line that looks like this:

```
# include CarrierWave::MiniMagick
```

Remove the `#` sign.

Below the line you just changed, add:

```
version :thumb do
  process :resize_to_fill => [50, 50]
end
```

The images uploaded from now on should be resized, but the ones we already have weren't affected. So edit one of the existing ideas and re-add a picture.

Displaying the thumbnails

To see if the uploaded picture was resized open `app/views/ideas/index.html.haml`. Change the line

```
%td= idea.picture
```

to

```
%td= image_tag idea.picture_url(file:///Users/klandrus/code/ruby-sho
```

Take a look at the list of ideas in the browser to see if the thumbnail is there.

Add design using HTML & CSS

Now the app is running well, but it still looks like scaffold. Let’s add some design to make it look like a professional website.

Adjust the application layout

First, let’s use bootstrap.min.css to apply a more lightweight style set to your app.

Open `app/views/layouts/application.html.haml` in your text editor and replace the line

```
<link rel="stylesheet" href="http://railsgirls.com/assets/bootstrap
```

with

```
<link rel="stylesheet" href="http://netdna.bootstrapcdn.com/twitter-
```

Open `app/assets/stylesheets/application.scss`, replace the line

```
body { padding-top: 100px; }
```

with

```
body { padding-top: 60px; }
```

Finally, delete the file `app/assets/stylesheets/scaffolds.css.scss` because we don't really need the default style generated by Rails.

Now refresh the page at http://localhost:3000/ideas. You will not find much change but it's good preparation for the following steps.

Refine the navigation

Considering “idea” is the most important object in your app, we are going to put the “New Idea” button on the navigation bar to make it always available.

Open `app/views/layouts/application.html.haml`, under the line

```
<li class="active"><a href="/ideas">Ideas</a></li>
```

add

```
%li= link_to 'New Idea', new_idea_path
```

Design the idea list

Now it's time to make the idea list page look professional. For that, we are going to replace the table layout with a div layout.

Open `app/views/ideas/index.html.haml` in your text editor and replace all lines with

```
%h1 Listing ideas
- @ideas.in_groups_of(3) do |group|
  .row
    - group.compact.each do |idea|
      .span4
        = image_tag idea.picture_url, :width => '100%' if idea.pictu
        %h4= link_to idea.name, idea
        = idea.description
```

Coach: Explain what the new code means line by line, and talk a little about Bootstrap 12 grids layout.

Refresh it! We get a nice looking idea list. Click the “New Idea” button, and create more ideas with real text and pretty pictures - the page will look much better with content. There is a principle of contemporary web design: content is the best decoration.

4.Design the idea details page

Click the title of an idea, and you will be brought to the details page of the idea. Now it is still scaffold generated by Rails, so let’s make it better.

Open `app/views/ideas/show.html.haml` in your text editor and replace all lines with

```
%p#notice= notice

.row
  .span9
    = image_tag(@idea.picture_url, :width => "100%") if @idea.pictu
  .span3
    %p
      %bName:
      = @idea.name
    %p
      %bDescription:
      = @idea.description
    %p
      = link_to 'Edit', edit_idea_path(@idea)
      \|
      = link_to 'Destroy', @idea, confirm: 'Are you sure?', method:
      \|
      = link_to 'Back', ideas_path
```

What next?

Use your new knowledge to design the new idea form Add more design to the pages as you wish Follow the other guides to learn more about Rails