

Ruby on Rails

Full Steam Ahead

by Danny Whalen & Brian Sam-Bodden

In Part 4 we'll explore Ajax in Rails

Objectives

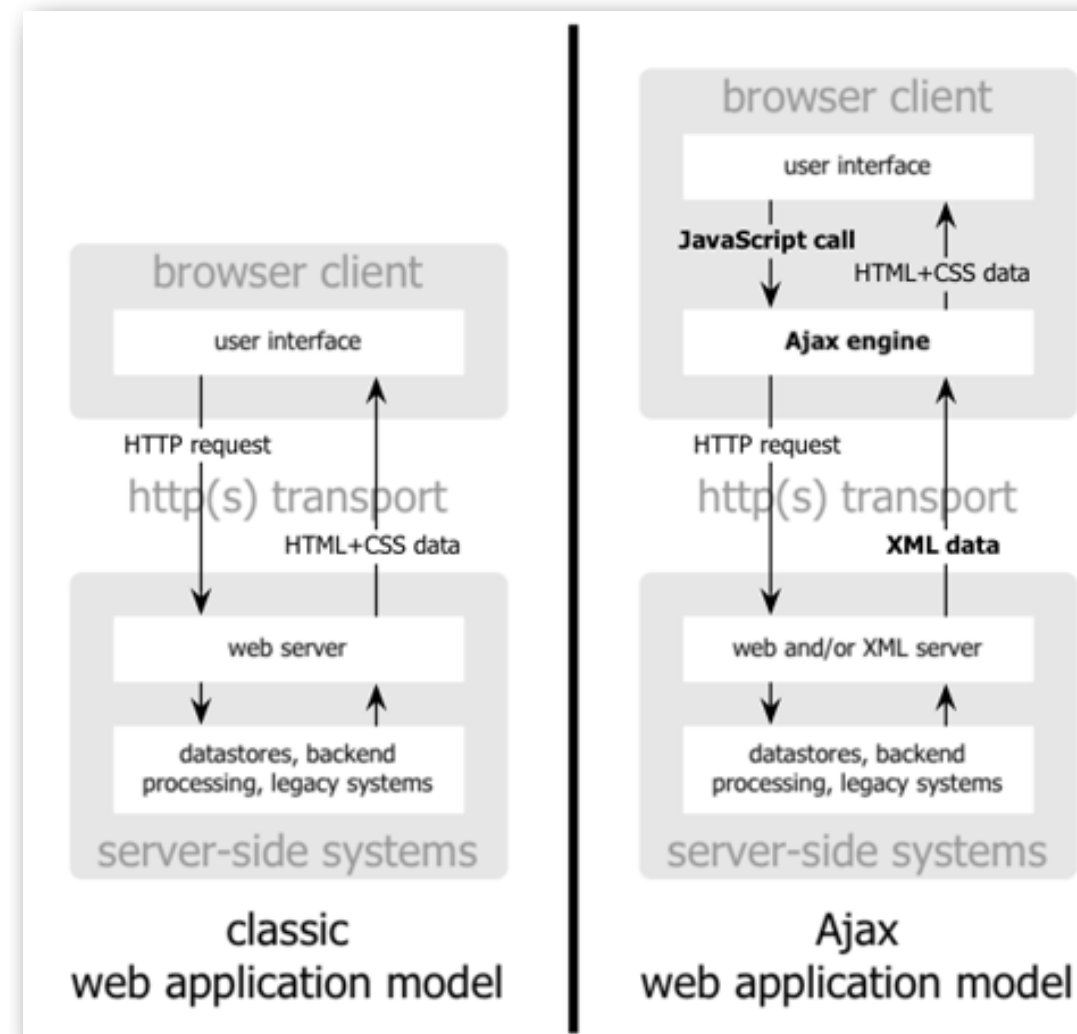
- Understand the basics of AJAX applications
- Understand how Rails implements AJAX

AJAX

Dynamic Rails

- AJAX stands for Asynchronous JavaScript And XML
- AJAX refers to any JavaScript technique for asynchronous interactions with the server using XML
- The foundations for AJAX were championed by Microsoft back in 1998 (using ActiveX as part of OWA*)
- Eventually, the non-MS browsers implemented a standard way encapsulated in the XHR (XMLHttpRequest) object
- A more loose definition of AJAX is any operation that updates a page without a full page refresh using data obtained from a remote call

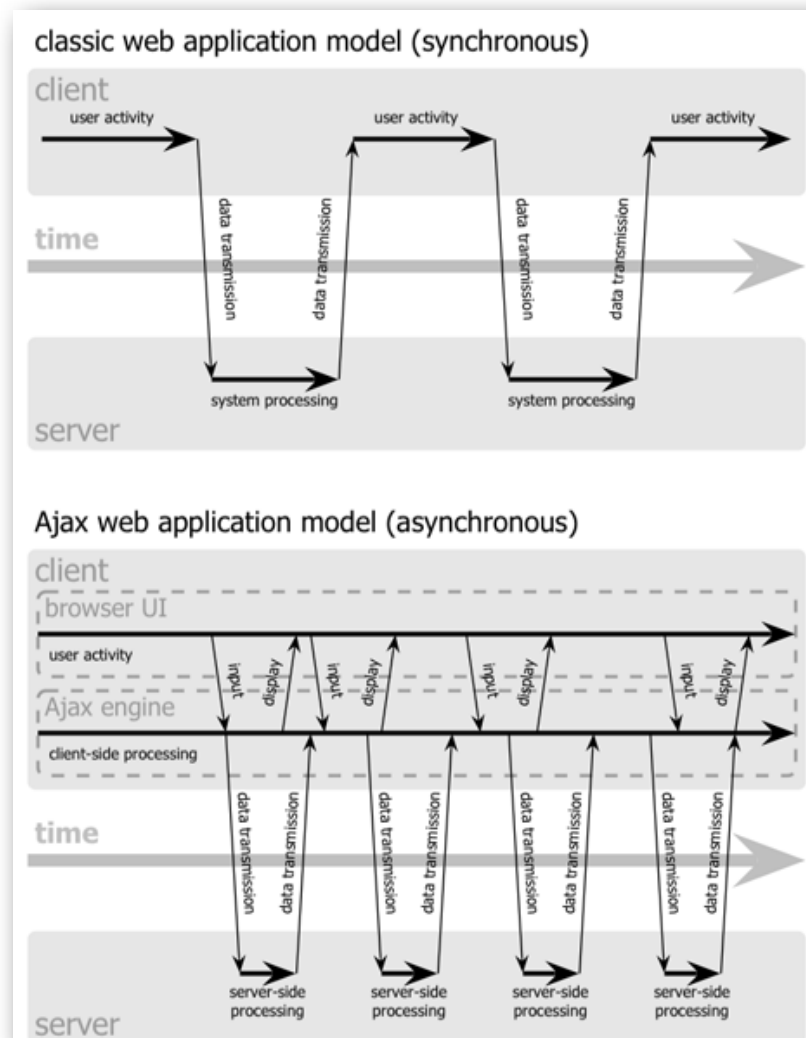
- AJAX (formal) application model, server returns XML, page is not refreshed and changes are applied dynamically with JS



AJAX

Dynamic Rich Web Pages

- In an AJAX application we typically see more and smaller requests, resulting in “chattier” applications



- AJAX stands for Asynchronous JavaScript And XML
- AJAX refers to any JavaScript technique for asynchronous interactions with the server using XML
- The foundations for AJAX were championed by Microsoft back in 1998 (using ActiveX as part of OWA*)
- Eventually, the non-MS browsers implemented a standard way encapsulated in the XHR (XMLHttpRequest) object
- A more loose definition of AJAX is any operation that updates a page without a full page refresh using data obtained from a remote call

AJAX in Rails

Dynamic Rich Web Pages

- Rails 3 implements all of its JavaScript Helper functionality (AJAX submits, confirmation prompts, etc) unobtrusively by adding the following HTML 5 custom attributes to HTML elements
 - data-remote: if true, submit via AJAX.
 - data-method: the REST method to use in form submissions
 - data-confirm: JS confirmation message before action
 - data-disable-with: disables form elements during a form submission

AJAX on Rails

Dynamic Rails

- Starting with Rails 3.1 ships with JQuery as the default JS library
- The prototype framework is still available if preferred
- By default a new Rails applications has the jquery-rails gem included
- In the application.js manifest both jquery and jquery_ujs (Unobtrusive JavaScript files are included

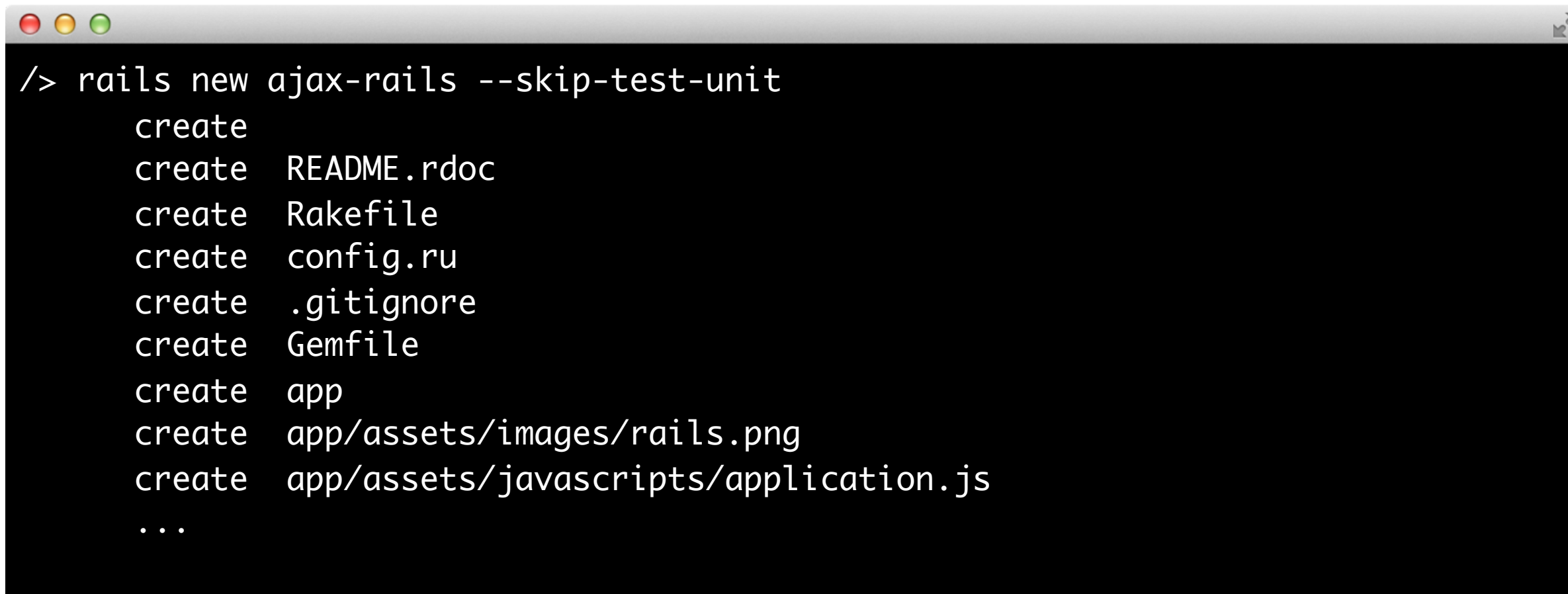
```
//= require jquery  
//= require jquery_ujs
```

app/assets/javascripts/application.js

```
gem 'jquery-rails'
```

Gemfile

- Let's create a simple Rails application that will server as the basis for our example

A terminal window with a dark background and light green text. The window has a title bar with three colored buttons (red, yellow, green) on the left and a close button on the right. The text inside the terminal shows the command to create a new Rails application and the files it creates.

```
/> rails new ajax-rails --skip-test-unit
create
create  README.rdoc
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
create  app
create  app/assets/images/rails.png
create  app/assets/javascripts/application.js
...
```

- We'll call the application ajax-rails and we'll skip Test::Unit based tests

- Let's also add a .rvmrc file:

A terminal window with a dark background and a light gray title bar. It contains the command to create a .rvmrc file.

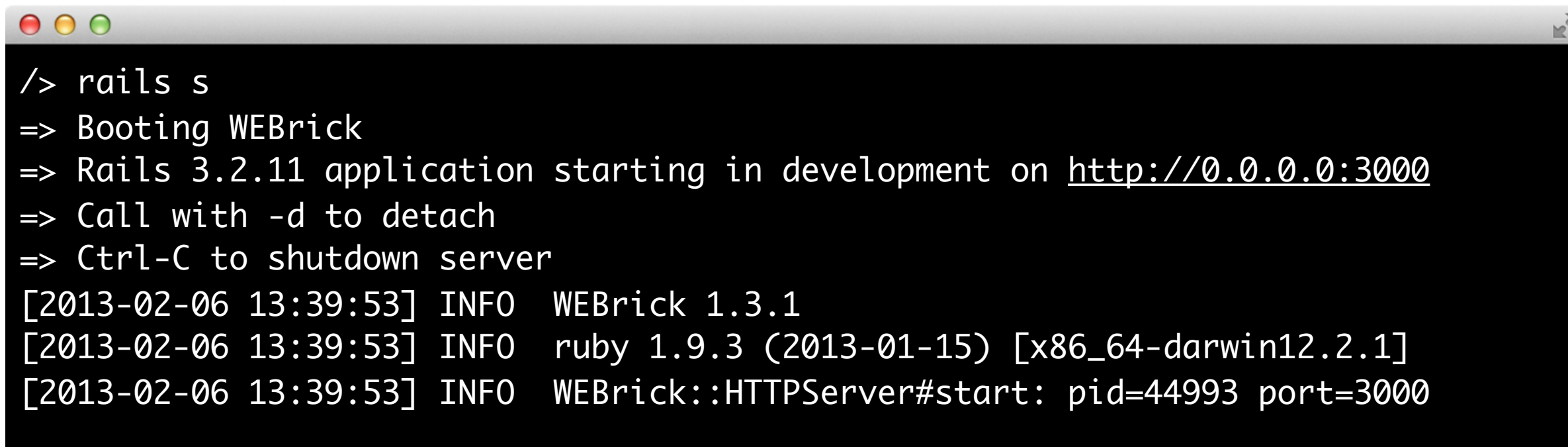
```
/> echo "rvm use 1.9.3@ajax-rails --create" > ajax-rails/.rvmrc
```

- Next we'll change directories to the application root folder, bundle the app and as customary we'll delete the index.html file:

A terminal window with a dark background and a light gray title bar. It contains three commands to change directory, bundle, and delete a file.

```
/> cd ajax-rails
/> bundle
/>rm public/index.html
```

- With a basic application in place we can run the server:

A screenshot of a terminal window with a dark background and white text. The window has a title bar with three colored buttons (red, yellow, green) on the left and a maximize button on the right. The text inside the terminal shows the command to start the Rails server and the subsequent output messages.

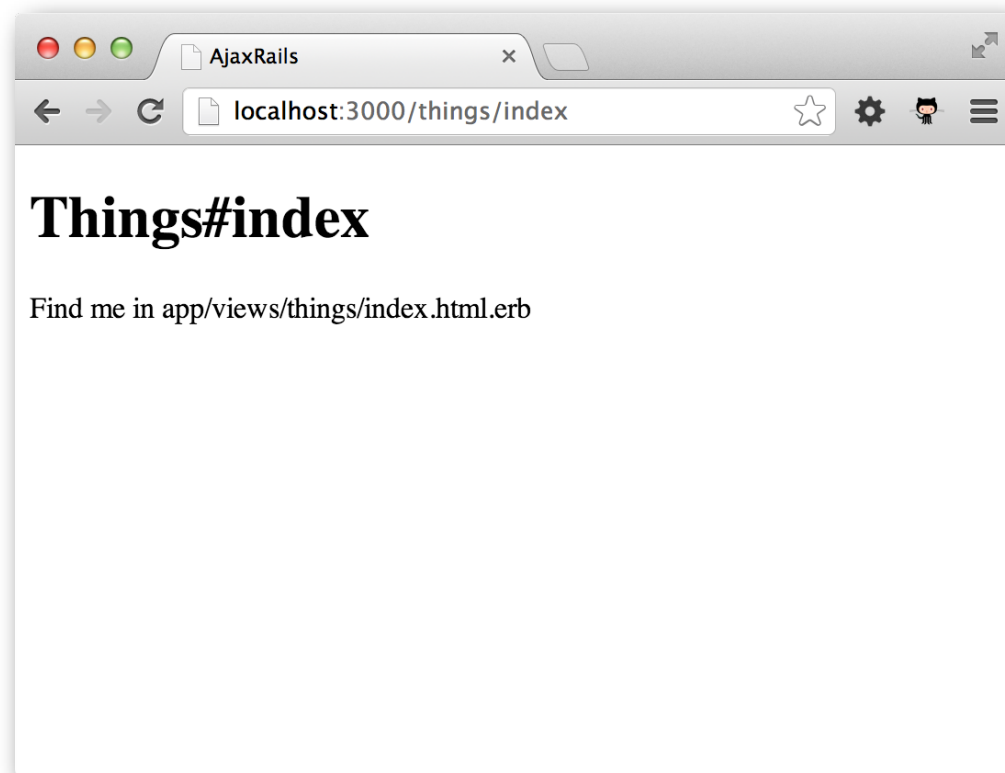
```
/> rails s
=> Booting WEBrick
=> Rails 3.2.11 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2013-02-06 13:39:53] INFO WEBrick 1.3.1
[2013-02-06 13:39:53] INFO ruby 1.9.3 (2013-01-15) [x86_64-darwin12.2.1]
[2013-02-06 13:39:53] INFO WEBrick::HTTPServer#start: pid=44993 port=3000
```



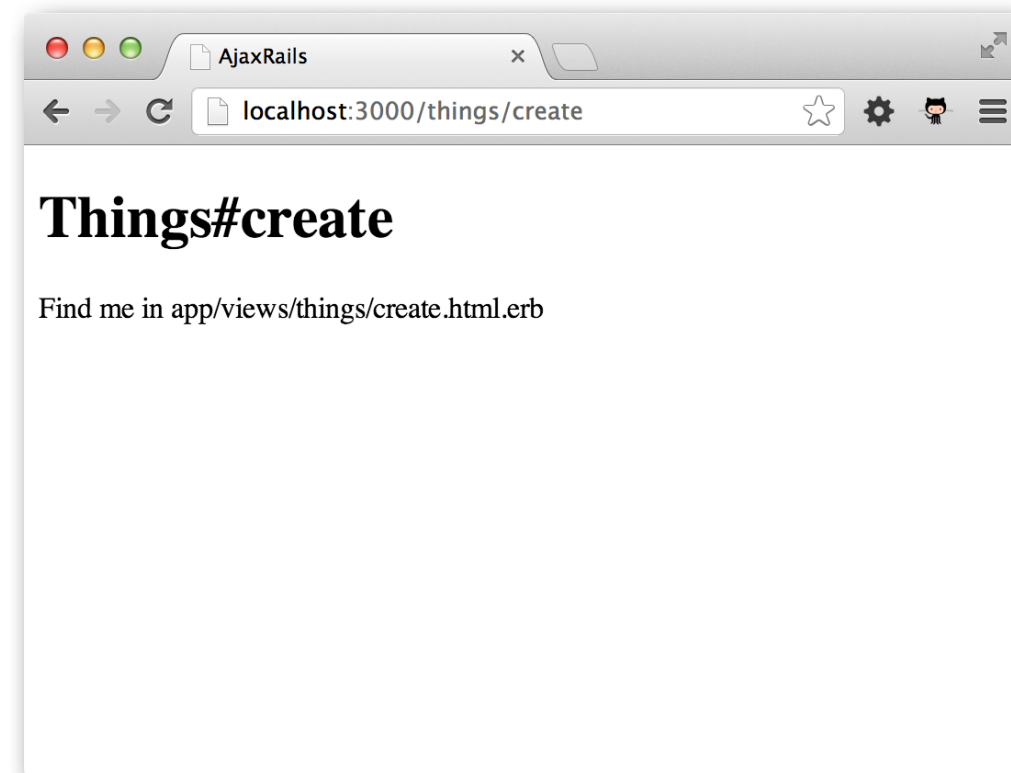
- We're going to start with the simplest AJAX example possible:
 - A link that will render some content dynamically on the page
- Let's create a model-less controller with two actions:

```
/>rails g controller things index create
  create  app/controllers/things_controller.rb
   route  get "things/create"
   route  get "things/index"
  invoke  erb
   create  app/views/things
   create  app/views/things/index.html.erb
   create  app/views/things/create.html.erb
  invoke  helper
   create  app/helpers/things_helper.rb
  invoke  assets
  invoke  coffee
   create  app/assets/javascripts/things.js.coffee
  invoke  scss
   create  app/assets/stylesheets/things.css.scss
```

- We can see the two actions at <http://localhost:3000/things/index> and <http://localhost:3000/things/create>



#index action



#create action

- Let's add a little bit of markup to the things/index.html.erb view:

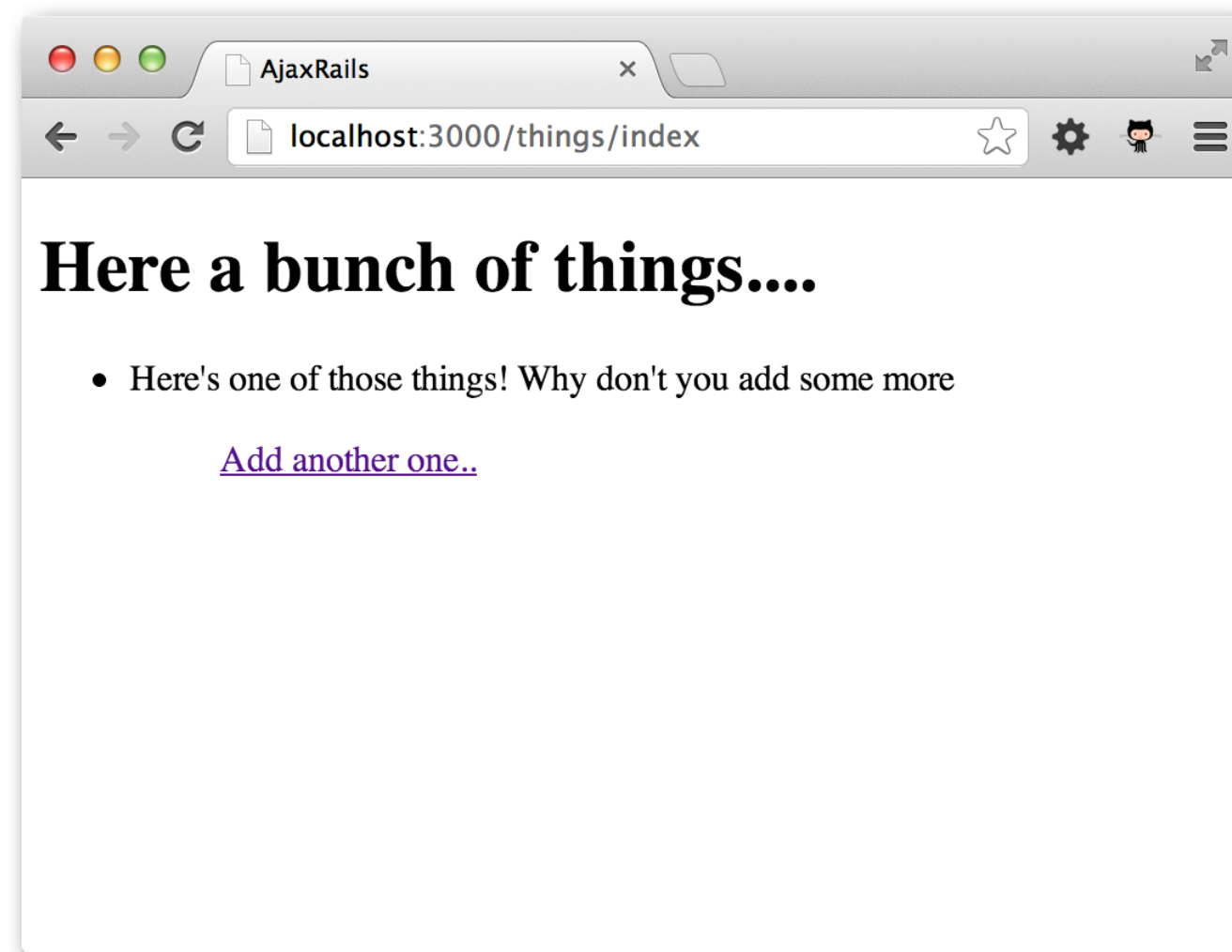
```
<h1>Here's a bunch of things....</h1>
<ul>
  <li>Here's one of those things! Why don't you add some more</li>
</ul>

<br/>

<%= link_to "Add another one..", :action => :create, :remote => true %>
```

app/views/things/index.html.erb

- Let's browse to the things#index action:



- Let's take a look at the generated markup:

```
<!DOCTYPE html>
<html>
<head>
  <title>AjaxRails</title>
  <link href="/assets/application.css?body=1" media="all" rel="stylesheet" type="text/css" />
  <link href="/assets/things.css?body=1" media="all" rel="stylesheet" type="text/css" />
  <script src="/assets/jquery.js?body=1" type="text/javascript"></script>
  <script src="/assets/jquery_ujs.js?body=1" type="text/javascript"></script>
  <script src="/assets/things.js?body=1" type="text/javascript"></script>
  <script src="/assets/application.js?body=1" type="text/javascript"></script>
  <meta content="authenticity_token" name="csrf-param" />
  <meta content="ceiW5LPM5vrVaJWUHBhjpK9tGmlidCCiXnDsLo64Gc4=" name="csrf-token" />
</head>
<body>

<h1>Here's a bunch of things....</h1>
<ul>
  <li>Here's one of those things! Why don't you add some more</li>
</ul>

<br/>
<a href="/things/create" data-remote="true">Add another one..</a>
</body>
</html>
```



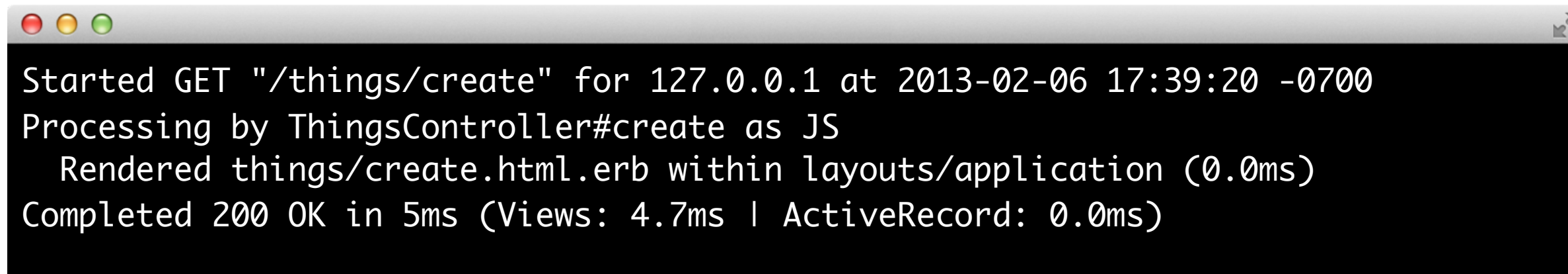
- The generated controller has two empty actions
- By default Rails will return an HTML view matching the name of the action

```
class ThingsController < ApplicationController
  def index
  end

  def create
  end
end
```

app/controllers/things/things_controller.rb

- If we click our link and look at the request on the running server console...

A screenshot of a terminal window with a dark background and white text. The window has a title bar with three colored buttons (red, yellow, green) on the left and a maximize button on the right. The text inside the terminal shows a GET request for "/things/create" from 127.0.0.1 at 2013-02-06 17:39:20 -0700, processed by ThingsController#create as JS, rendered things/create.html.erb within layouts/application (0.0ms), and completed 200 OK in 5ms (Views: 4.7ms | ActiveRecord: 0.0ms).

```
Started GET "/things/create" for 127.0.0.1 at 2013-02-06 17:39:20 -0700
Processing by ThingsController#create as JS
  Rendered things/create.html.erb within layouts/application (0.0ms)
Completed 200 OK in 5ms (Views: 4.7ms | ActiveRecord: 0.0ms)
```

- ... we can see that the request is a GET and that it is being processed by the ThingController as JS (JavaScript)
- ... since we don't have a JS response, Rails default behavior is to return the existing HTML template response (which won't have any effect on our page)

- Let's remedy that by modifying our controller to return JS as follows:

```
class ThingsController < ApplicationController
  def index
  end

  def create
    respond_to do |format|
      format.js {}
    end
  end
end
```

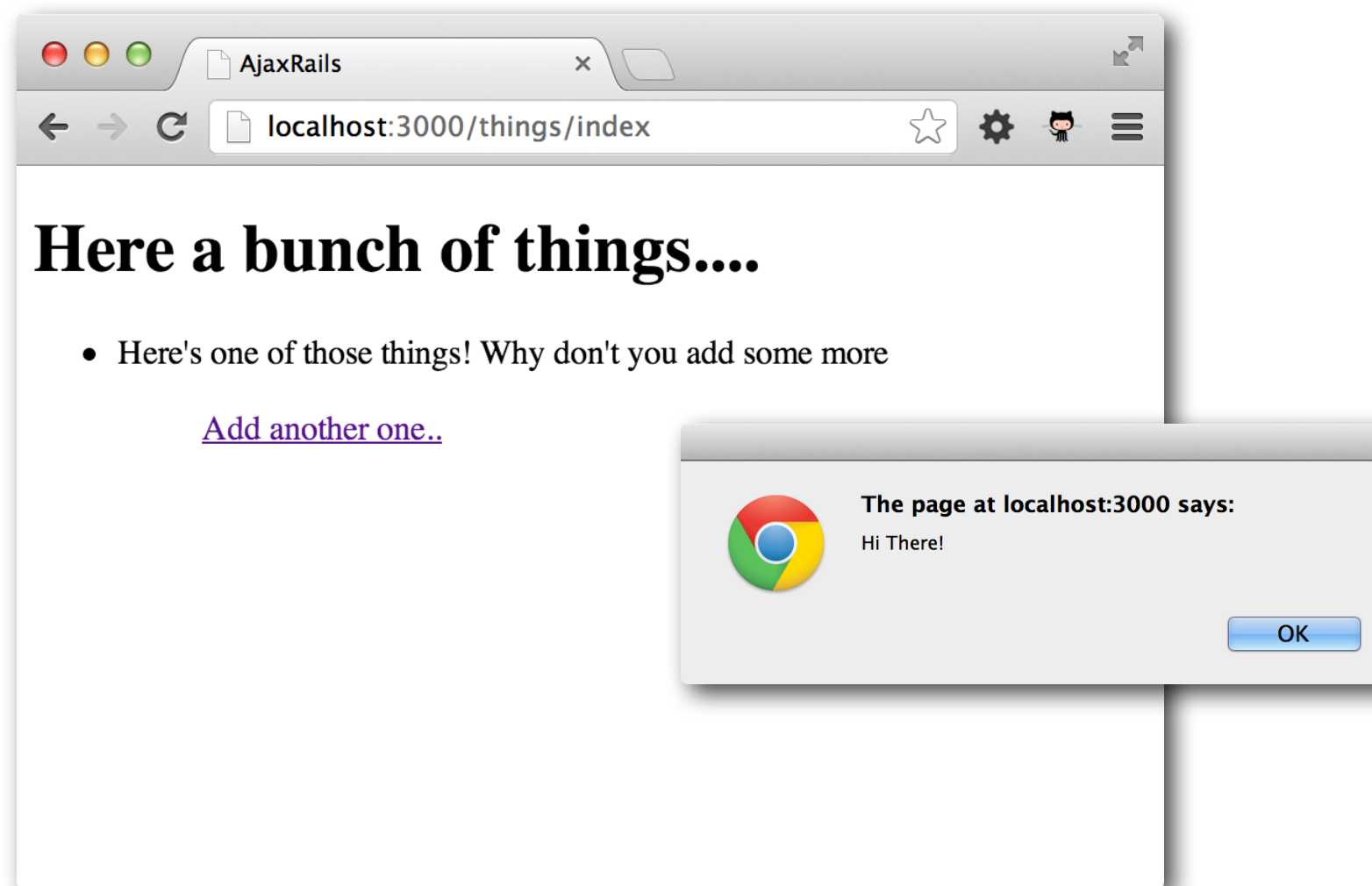
app/controllers/things/things_controller.rb

- We also need a proper JS response “view” to send to the client
- Let's rename the create.html.erb to create.js.erb and change the contents to:

```
alert("Hi There!");
```

```
app/views/things/create.js.erb
```

- We can now test our UI, clicking on the link should bring up the JS dialog
- The returned JS response is EoA (Executed on Arrival) at the browser





- If we check the request/response on the server output we can see that the JS response was rendered:

```
Started GET "/things/create" for 127.0.0.1 at 2013-02-06 19:03:17 -0700
Processing by ThingsController#create as JS
  Rendered things/create.js.erb (0.0ms)
Completed 200 OK in 3ms (Views: 3.1ms | ActiveRecord: 0.0ms)
```


- Now that we have the plumbing working let's modify the markup and add an id to the UL so that we can do some interesting things with/to it:

```
<h1>Here a bunch of things....</h1>
<ul id="list-of-things">
  <li>Here's one of those things! Why don't you add some more</li>
</ul>

<br/>

<%= link_to "Add another one..", :action => :create, :remote => true %>
```

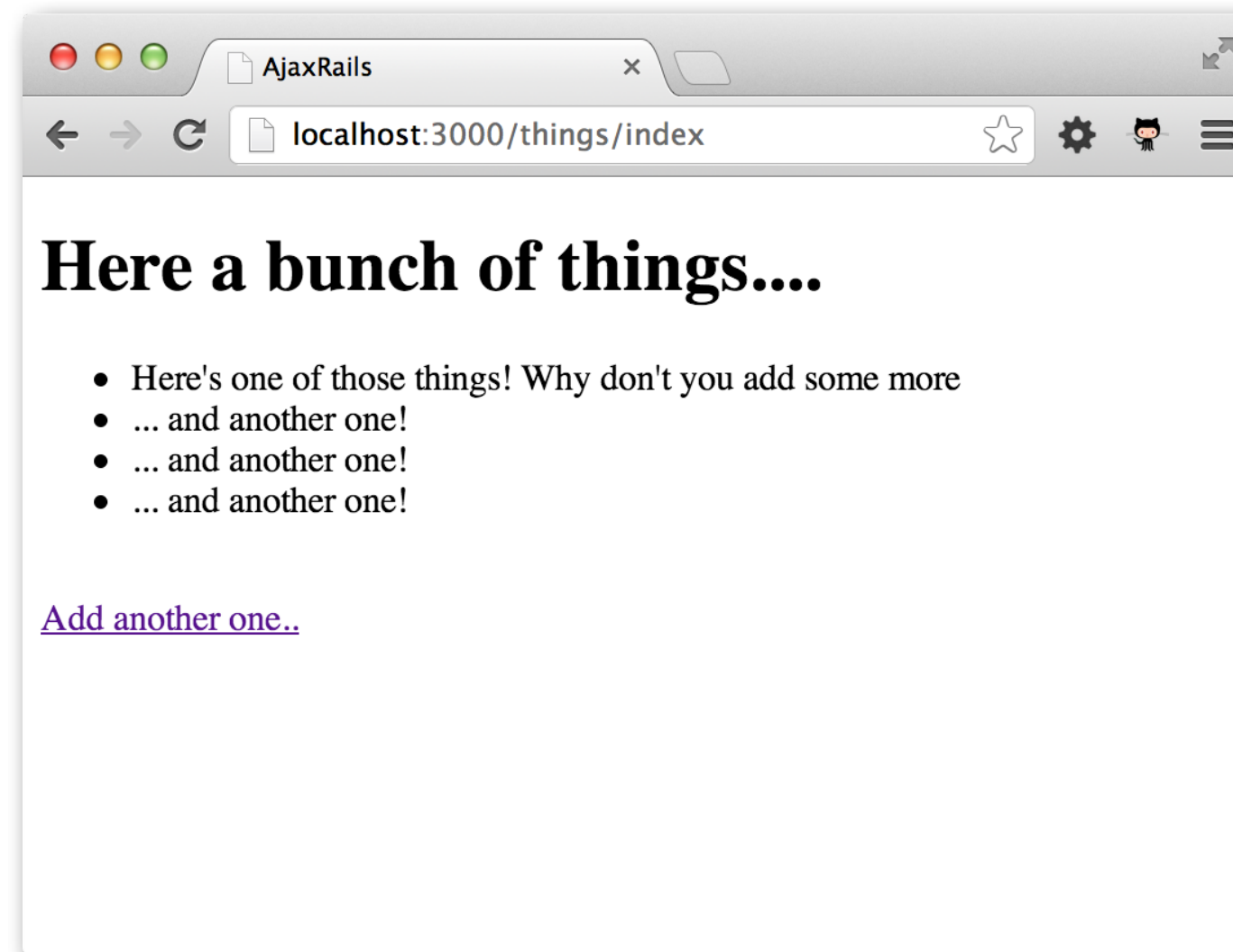
app/views/things/index.html.erb

- The id "list-of-things" on the markup we can now rewrite our JS snippet to find that and append a new to it:

```
$('#list-of-things').append("<li>... and another one!</li>");
```

```
app/views/things/create.js.erb
```

- Now, clicking the link adds new lines to our list:



- In most cases we want the markup generated by the server to have something dynamic about it, let's refactor the controller to provide a random word and its definition in an instance variable `@funny_word`:

```
class ThingsController < ApplicationController
  ...
  def create
    @funny_word = FUNNY_WORDS.sample
    respond_to do |format|
      format.js {}
    end
  end

  private

  # http://www.alphadictionary.com/articles/100\_funniest\_words.html
  FUNNY_WORDS = [
    "Abibliophobia: The fear of running out of reading material.",
    "Absquatulate: To leave or abscond with something.",
    ...
  ]
end
```

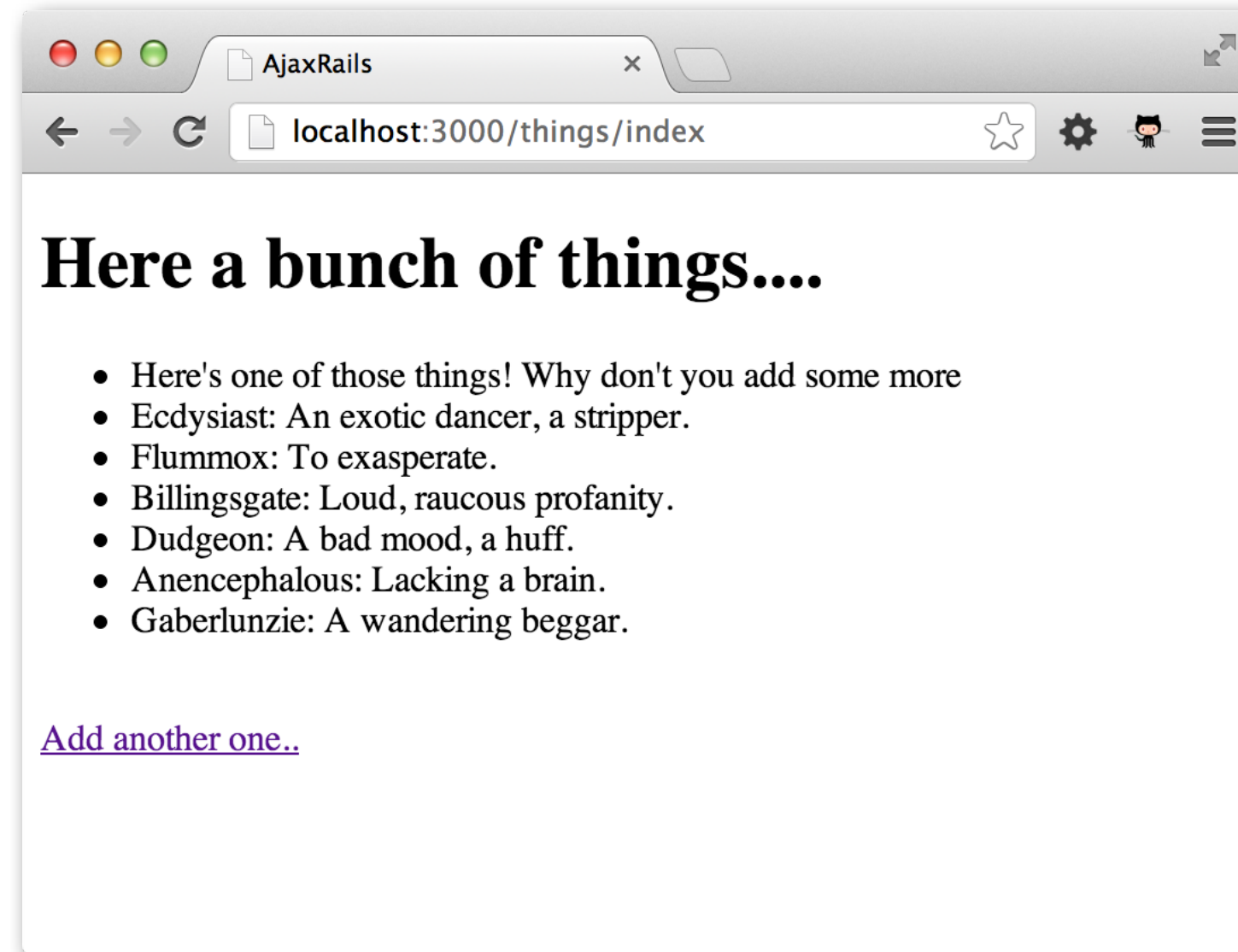
app/controllers/things/things_controller.rb

- Let's refactor the JS template to include (using ERB) our "funny word":

```
$('#list-of-things').append("<li><%= @funny_word %></li>");
```

```
app/views/things/create.js.erb
```

- Now, clicking the link should add a random funny word to the list:



- When working with AJAX it is convenient to have a browser with tools that can allow you inspect the request and responses:

