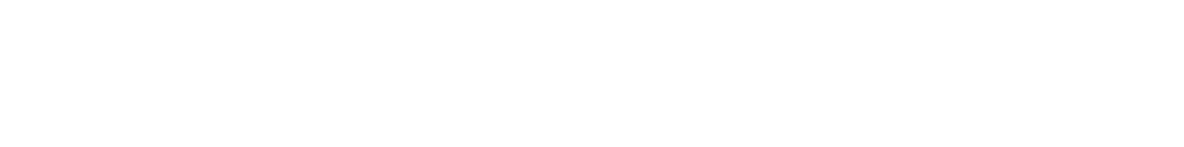
[**Follow**](#_bookmark0) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark0) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Creating a new app

Module 1. Getting started with Rails



[← Previous](#_bookmark0)

[Next →](#_bookmark1)

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0)  [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4)  [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6)  [Rails](#_bookmark6) |
| [After hello, say](#_bookmark8) ['greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11)  [controller](#_bookmark11) |
| [Home & about](#_bookmark13)  [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

[Module 2. Starting with to-do list app](#_bookmark0) [Module 3. Add, delete, mark &](#_bookmark0)

[unmark todos](#_bookmark0)

### How to have fun learning Rails?

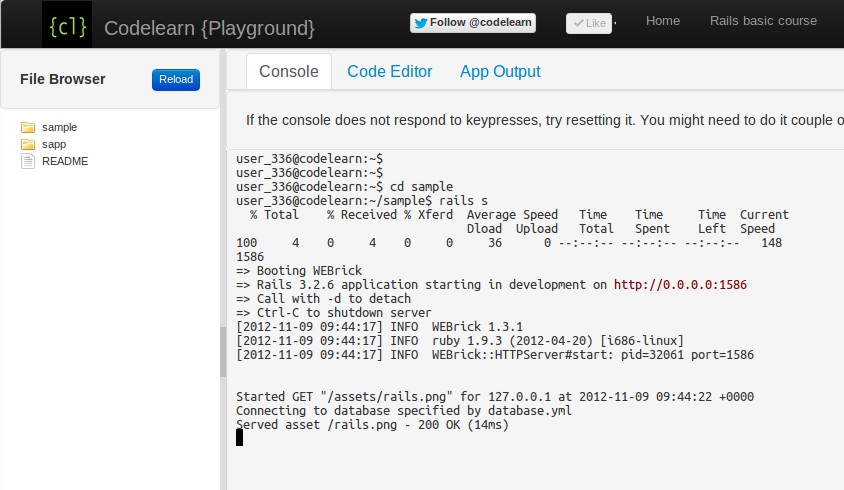
I hope you had a look at [Codelearn Playground](http://www.codelearn.org/apps/code_play). If you haven't, this is probably right time to have a sneak peek.

Codelearn Playground emulates the complete Rails stack inside browser. So you do not need to install anything on your PC to get started.

Follow the lessons, execute commands in the Console, browser files in File Browser, edit files in Code Editor & watch your rails app in action in App Output.

Figure: Screenshot of Console

This course is fun, if you simultaneously build (a rails website) as it is taught. This is not one of those boring books that you read like a novel. Its like one of those toys that you play with. The difference, you actually learn to build a rails



website here.

### Building your Rails app in 5 steps.

1. Ruby on Rails websites begin with the default app. The way to create it is to execute rails new your\_site inside the Console in [Codelearn Playground](http://www.codelearn.org/apps/code_play). The Rails app name here is your\_site.

Console

user\_1@codelearn:~$ rails new your\_site

Figure: Screenshot of rails new command output on Codelearn Playground console.

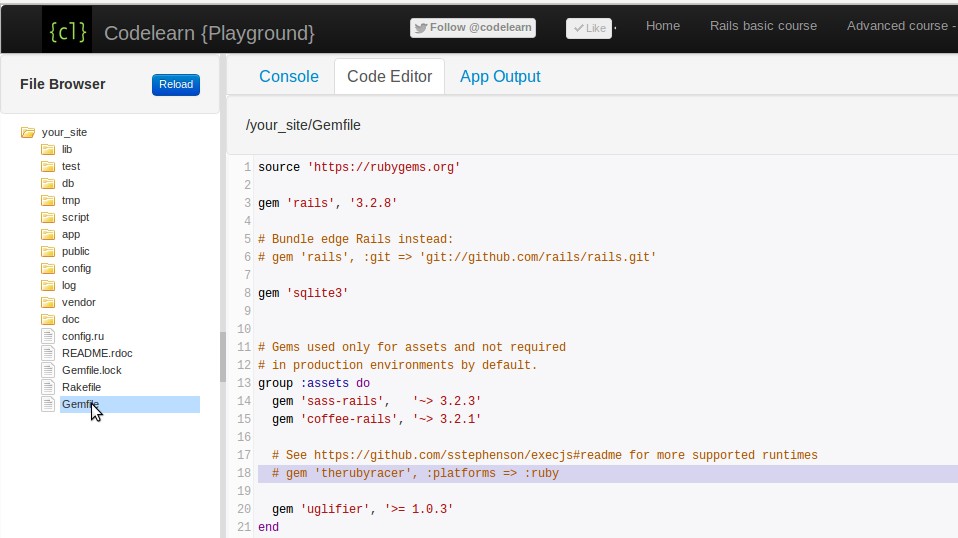
I had to paste the output here for reference. It is really long, sorry for that. When you do this step in the Console of Codelearn Playground, the bundle



install command is not executed. You would need to explicitly execute the command by getting into the directory from the console.

1. The rails new your\_site creates the Rails app in a new directory your\_site.

So lets get into the directory in Console.



Console

user\_1@codelearn:~$ cd your\_site user\_1@codelearn:~/your\_site$

1. Inside File Browser, click on Gemfile to view it in Code Editor and uncomment the line below.

Note In Ruby (and in rails) comments are lines, beginning with "#" hash. You uncomment the line by removing the "#".

Figure: Gemfile, and the line to uncomment.

Gemfile

* 1. gem 'therubyracer', :platforms => :ruby

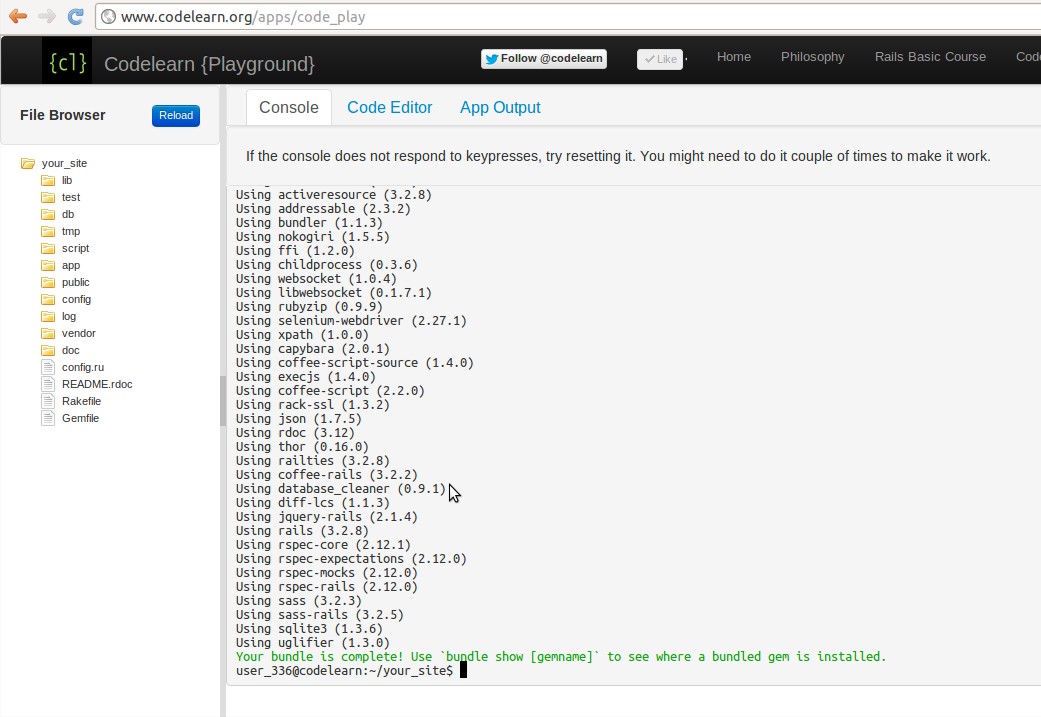
1. Save the Gemfile. Run bundle install in the Console. The below output is long, and it may be different in your case.

Console

user\_1@codelearn:~/your\_site$ bundle install

Figure: Screenshot of bundle install command output on Codelearn Playground console.

1. Execute rails server in Console.



Note "rails s" is short form for "rails server"

Console

user\_1@codelearn:~$ rails server

=> Booting WEBrick

=> Rails 3.2.6 application starting in development on http://0.0.0.0:3000

=> Call with -d to detach

=> Ctrl-C to shutdown server

[2012-08-18 19:45:32] INFO WEBrick 1.3.1

[2012-08-18 19:45:32] INFO ruby 1.9.3 (2012-04-20) [i686-linux]

[2012-08-18 19:45:32] INFO WEBrick::HTTPServer#start: pid=21368 port=3000

Once the server is running, go to the App Output tab & hit Reload button to see the output.

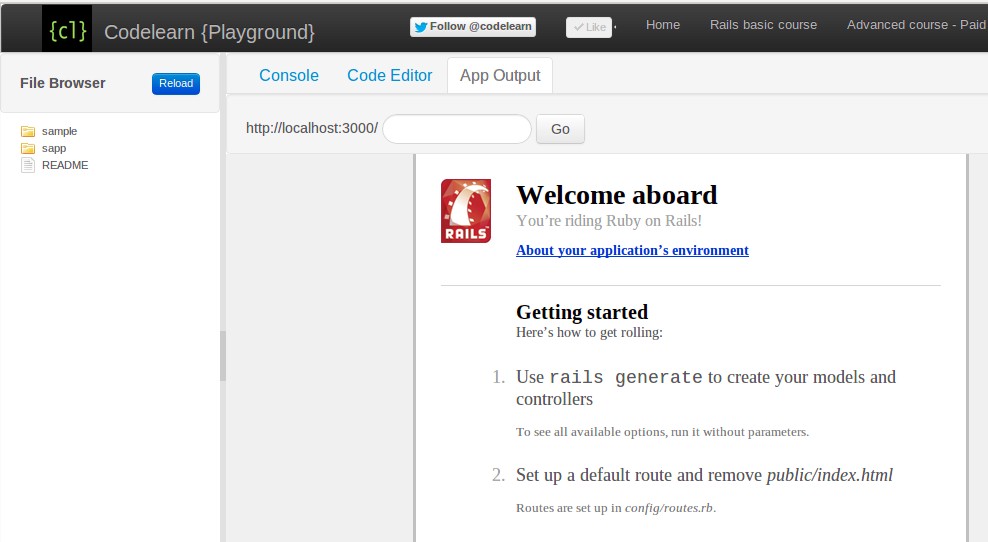


Figure: Screenshot of default Rails app in App Output tab

On a local Rails installation, the server started via rails server command will listen at port 3000 by default. Hence you would notice

- http://localhost:3000 - in the top bar of the App Output tab.

In our case, to avoid conflict of port when multiple users execute rails server command, we allocate a random port to every user.

### Common Errors

* 1. If you run rails server without uncommenting the therubyracer gem in the Gemfile. You will get the following error.

Remember to run "bundle install" everytime you change the Gemfile

Figure: Screenshot of rails server error without including therubyracer gem

* 1. Executing rails server or bundle install outside the rails app will not work.

Below is example of how it will look if rails server is executed outside the rails app. Note that the user is not inside your\_site while executing rails server command.

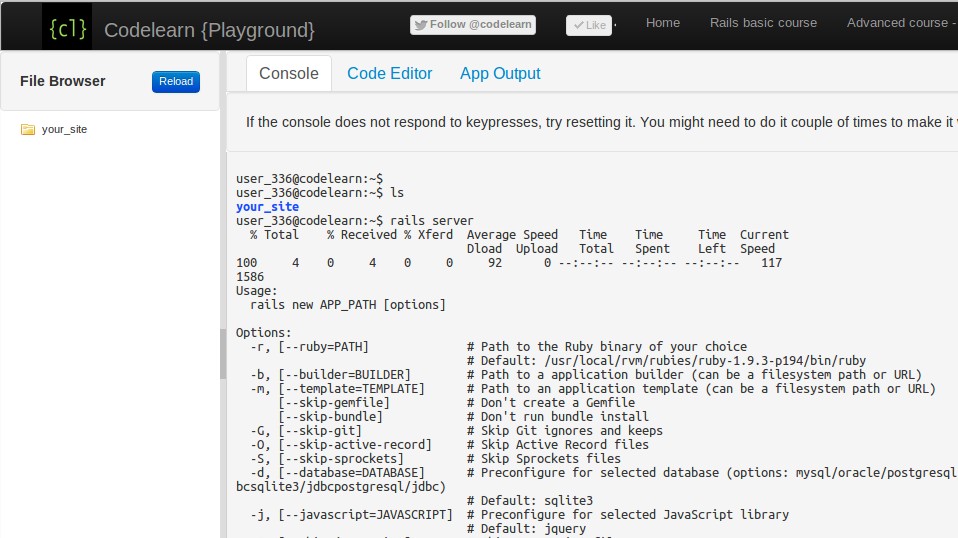


Figure: Error when you fire rails server outside the rails app directory.

[← Previous](#_bookmark0)

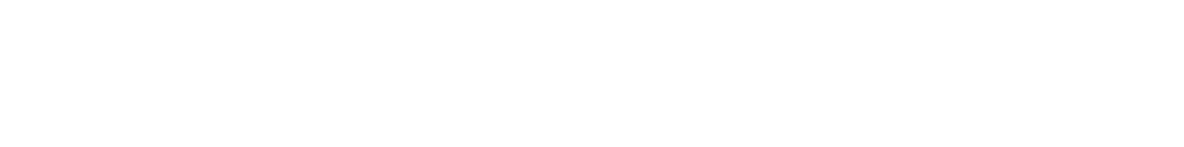
[Next →](#_bookmark1)

© Codelearn 2012

[**Follow**](#_bookmark1) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark1) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Create default app your\_site

[Module 1. Getting started with Rails](#_bookmark2)



[← Previous](#_bookmark0) [Next →](#_bookmark4)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.



|  |
| --- |
| [1. Zero to app in 5](#_bookmark0)  [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4)  [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6)  [Rails](#_bookmark6) |
| [After hello, say 'greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13) [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)



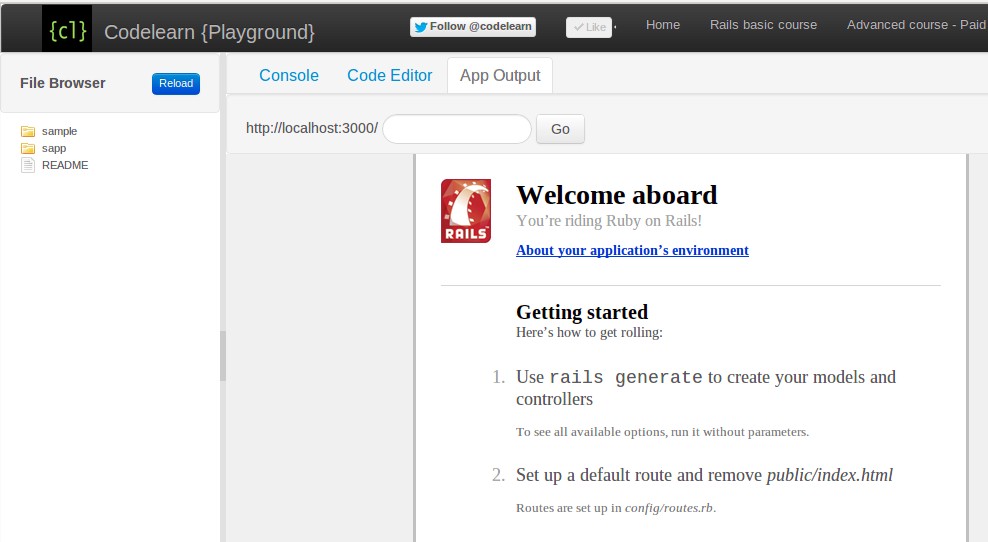
1. Follow instructions in the [parent lesson](#_bookmark0) to create a new rails app with the name your\_site.



1. Inside Codelearn Playground, execute rails server in the Console, navigate to App Output & the app should show as below.



[Module 2. Starting with to-do list app](#_bookmark1)



[Module 3. Add, delete, mark &](#_bookmark1) [unmark todos](#_bookmark1)

Figure: Screenshot of default rails app in App Output

Test Output

Click 'Run test' button above to verify if your task solution is right.

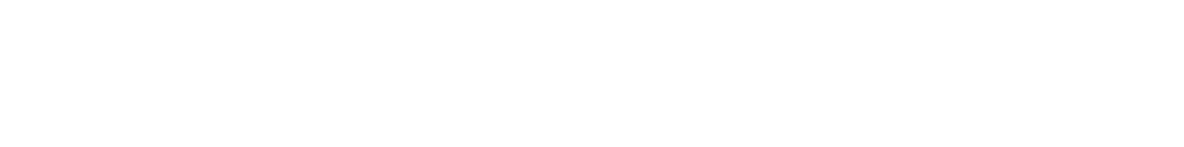
[← Previous](#_bookmark0) [Next →](#_bookmark4)

© Codelearn 2012

[**Follow**](#_bookmark4) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark4) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Understanding Rails directory

[Module 1. Getting started with Rails](#_bookmark5)



[← Previous](#_bookmark1) [Next →](#_bookmark6)

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0)  [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4)  [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6)  [Rails](#_bookmark6) |
| [After hello, say](#_bookmark8) ['greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11)  [controller](#_bookmark11) |
| [Home & about](#_bookmark13)  [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

[Module 2. Starting with to-do list app](#_bookmark4) [Module 3. Add, delete, mark &](#_bookmark4)

[unmark todos](#_bookmark4)

The [last lesson](#_bookmark0) showed how to create a default rails app in 5 steps inside Codelearn Playground.

This lesson shows the directory structure of the newly created app & which files are important in it.

The good thing about Rails is that all Rails apps are similar, every file has a place and the place doesn't change no matter which app you are looking at.

The not so good thing is that the rails default app has too many of files & directories. In this lesson, we will find out which ones are important.

### The 4 most important files and directories.

1. public/index.html is the file that gets rendered when you see the output of the default Rails app in App Output tab. Other publicly accessible files resides here. Rails do not process these files & they are accessible as is from the browser.



1. Gemfile - The list of gems (Rails libraries) needed for your Rails app resides here.

Figure: Rails app directory structure.

Note Any inclusion/exclusion of gems in the file would need you to run bundle install again for the changes to take effect.

1. app/ directory contains all app related files - the models, controllers and views. This is the heart of your app where all

the magic happens. More about these in the subsequent modules.

1. config/ directory contains files related to the app configuration.

Note Any changes in the configuration file would require you to restart Rails server for the changes to take effect.

### 3 important things about Gemfile.

1. A gem is a self-contained module of ruby code. Somebody else has created it and you use it. It is like a self contained external library.
2. When you run bundle install, it goes through the gems listed in Gemfile one by one, fetches them, installs them locally & include them in the app.
3. There is version information that can be optionally specified with every gem. For eg. gem 'rails', '3.2.6' instructs bundle install to fetch rails version 3.2.6



Figure: A typical Gemfile

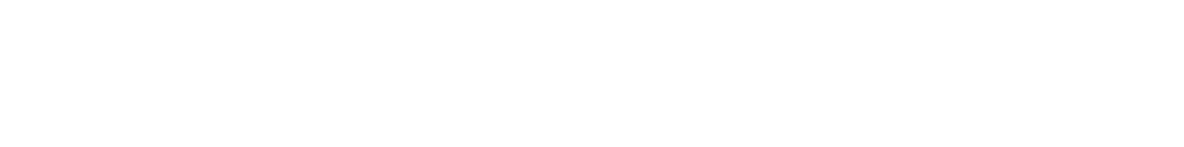
[← Previous](#_bookmark1) [Next →](#_bookmark6)

© Codelearn 2012

[**Follow**](#_bookmark6) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark6) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Adding Helloworld page

[Module 1. Getting started with Rails](#_bookmark7)



[← Previous](#_bookmark4) [Next →](#_bookmark8)

You looked into the directory structure of the default Rails app in the [last](#_bookmark4) [lesson](#_bookmark4).

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0) [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4) [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6)  [Rails](#_bookmark6) |
| [After hello, say 'greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13)  [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |



In this lesson, you will create a static page HelloWorld.html in Rails.



1. Create a new file public/HelloWorld.html. You can create a new file in the public folder as follows.



Note You might have rails server running inside the Console. You can end the process by pressing Control+C.



Console



user\_1@codelearn:~/your\_site$ cd public user\_1@codelearn:~/your\_site/public$ touch HelloWorld.html



2. Locate the file HelloWorld.html in File Browser, click on it to see its content in File Editor tab. The file will be empty. Populate the file with the lines below. public/HelloWorld.html



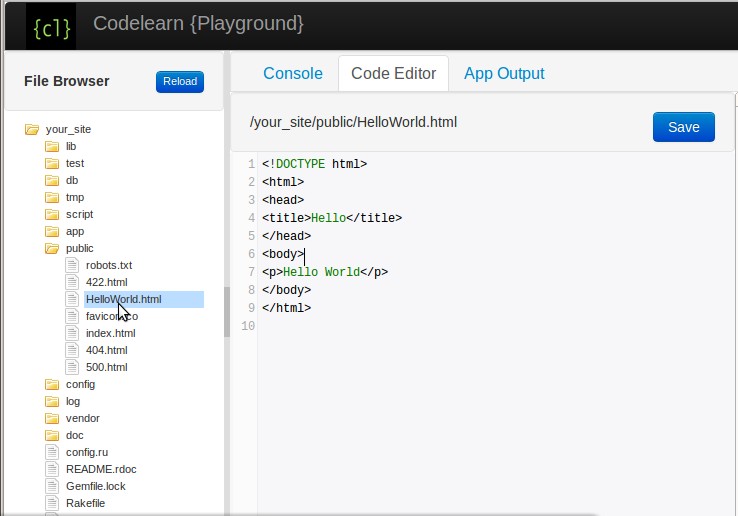
[Module 2. Starting with to-do list app](#_bookmark6) [Module 3. Add, delete, mark &](#_bookmark6)

[unmark todos](#_bookmark6)

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>Hello</title>
5. </head>
6. <body>
7. <p>Hello World</p>
8. </body>
9. </html>

Figure: How the file public/HelloWorld.html look after population

* 1. Come out of the public directory to your\_site. Run rails server.



Console

user\_1@codelearn:~/your\_site/public$ cd .. user\_1@codelearn:~/your\_site$ rails server

* 1. Navigate to App Output tab & enter 'Helloworld.html' (without the quotes, the file name is case sensitive) in the input field next to http://localhost:3000 & press Go button to view the newly created HelloWorld page.

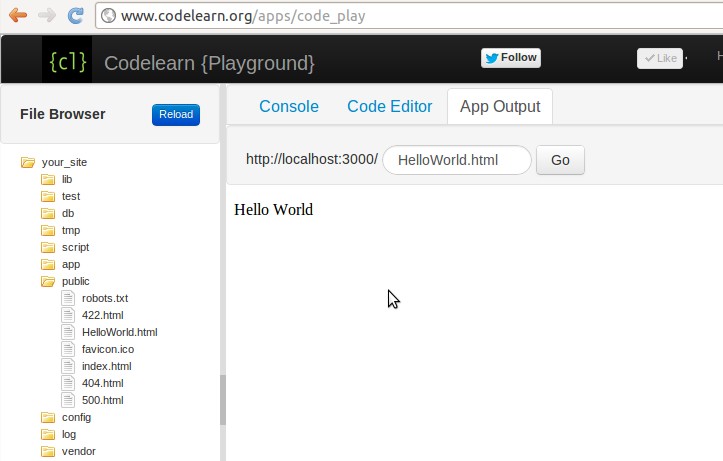


Figure: Helloworl Output

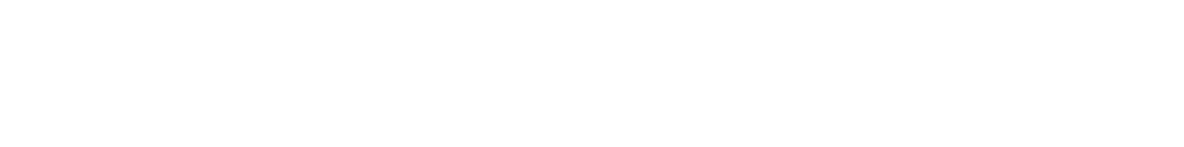
[← Previous](#_bookmark4) [Next →](#_bookmark8)

© Codelearn 2012

[**Follow**](#_bookmark8) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark8) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Create greetings page

[Module 1. Getting started with Rails](#_bookmark9)



[← Previous](#_bookmark6) [Next →](#_bookmark11)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.



|  |
| --- |
| [1. Zero to app in 5](#_bookmark0)  [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4)  [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6)  [Rails](#_bookmark6) |
| [After hello, say 'greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13) [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)



Start with the default Rails app created in [the task of Lesson 1](#_bookmark1).



Follow the parent lesson - [Say hello through Rails](#_bookmark6) - to understand how the hello world page is created.



On similar lines, create a page public/Greetings.html

Populate Greetings.html with the following code.



[Module 2. Starting with to-do list app](#_bookmark8) [Module 3. Add, delete, mark &](#_bookmark8)

[unmark todos](#_bookmark8)

public/Greetings.html

1. <h1> Hello World</h1>

1. <p> Hello How are you? </p>
2. <br>
3. <a href="index.html"> Back </a>

Once done, start rails server inside Console and the new page should be visible at the URL localhost:3000/Greetings.html. To view it in Codelearn Playground, navigate to App Output tab, type Greentings.html in the input field & hit the Go button.

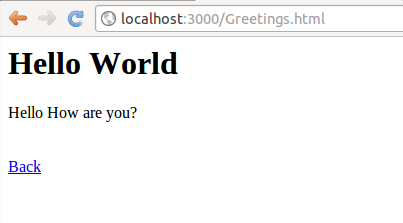


Figure: Screenshot of Greetings.html

Test Output

Click 'Run test' button above to verify if your task solution is right.

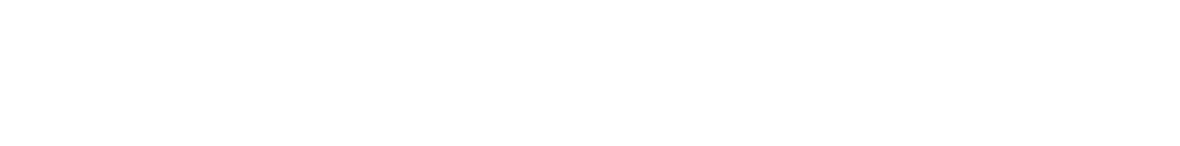
[← Previous](#_bookmark6) [Next →](#_bookmark11)

© Codelearn 2012

[**Follow**](#_bookmark11) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark11) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Adding pages from Rails controller

[Module 1. Getting started with Rails](#_bookmark12)



[← Previous](#_bookmark8) [Next →](#_bookmark13)

In the previous lesson - [Adding Helloworld page](#_bookmark6), you created a static page

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0) [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4) [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6) [Rails](#_bookmark6) |
| [After hello, say](#_bookmark8) ['greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13) [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

public/HelloWorld.html.

In this lesson, you will create some more pages but they will not be inside the public directory. You will learn to use the Rails controller to create them.

Lets say your your\_site is your website to which you like to add two pages home & about. Lets say the pages needs to be accessed as http://localhost:3000/pages/home & http://localhost:3000/pages/about

### Adding home & about pages through controller

In short & quick words, we will have a controller called Pages. home & about are two actions inside the controller. More on what a controller is and why it is

needed later.

[Module 2. Starting with to-do list app](#_bookmark11)

rails server might already running in your Console. You need to kill it to be

[Module 3. Add, delete, mark &](#_bookmark11) [unmark todos](#_bookmark11)

able to execute any more commands. You can kill it by pressing Ctrl+C.

Command

user\_1@codelearn:~/your\_site$ rails generate controller Pages home about

Command Output

create app/controllers/pages\_controller.rb route get "pages/about"

route get "pages/home" invoke erb

create app/views/pages

create app/views/pages/home.html.erb create app/views/pages/about.html.erb invoke test\_unit

create test/functional/pages\_controller\_test.rb invoke helper

create app/helpers/pages\_helper.rb

invoke test\_unit

create test/unit/helpers/pages\_helper\_test.rb invoke assets

invoke coffee

create app/assets/javascripts/pages.js.coffee invoke scss

create app/assets/stylesheets/pages.css.scss user\_1@codelearn:~/your\_site$

### What did "rails generate controller" do?

* 1. It created routes for the URLs.

config/routes.rb

* + 1. SampleApp::Application.routes.draw do
    2. get "pages/home"

3.

4. get "pages/about"

5.

* 1. It also created a controller file with two methods/funtions, also called actions, home & about.

app/controllers/pages\_controller.rb

* + 1. class PagesController < ApplicationController
    2. def home

3. end

4.

1. def about
2. end
3. end

Controller handles URL requests and decides what to do with it.

3. It also generated view files corresponding to the actions. For action home, the view file is app/views/pages/home.html.erb. Similarly for about, it is app/views/pages/about.html.erb.

The above files have extension .html.erb these are normal html files along with erb - embedded ruby. The extension implies that these files may contain ruby code. It allows for generating dynamic content.

There are other files that are generated , test files, scss files, js files etc which we would not bother for now.

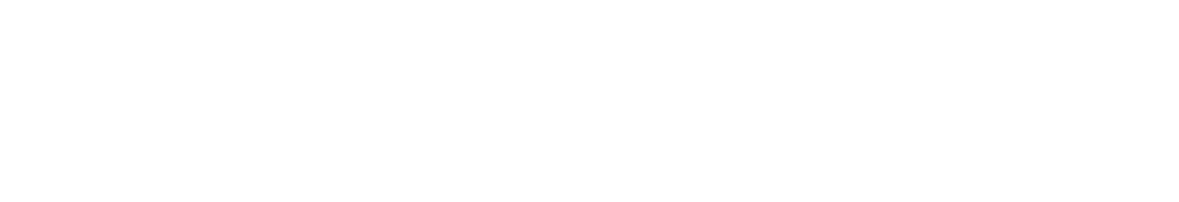
[← Previous](#_bookmark8) [Next →](#_bookmark13)

© Codelearn 2012

[**Follow**](#_bookmark13) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark13) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

Rails tutorial - Task - Create home & about pages through controller

[Module 1. Getting started with Rails](#_bookmark14)



[← Previous](#_bookmark11) [Next →](#_bookmark16)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0) [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4) [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6) [Rails](#_bookmark6) |
| [After hello, say](#_bookmark8) ['greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13) [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In this task, you need to create two pages accessible at the urls http://localhost:3000/pages/home & http://localhost:3000/pages/about.

The page content is simple placeholder content in the view files which rails generate controller command create. See the screenshot at the end of the task.

1. You need to have a default Rails app before you start to attempt this task. Follow the first lesson - [Zero to app in 5 steps](#_bookmark0) - to get the default app.

[Module 2. Starting with to-do list app](#_bookmark13) [Module 3. Add, delete, mark &](#_bookmark13)

[unmark todos](#_bookmark13)

1. Follow the parent lesson - [Pages using controller](#_bookmark11) - to get the two pages - home & about.
2. Like the earlier tasks, execute rails server in Console.
3. Navigate to App Output. Entering /pages/home & /pages/about one by one in the top input box next to 'http://localhost:3000' should show you the respective page content as shown in the screenshots below.

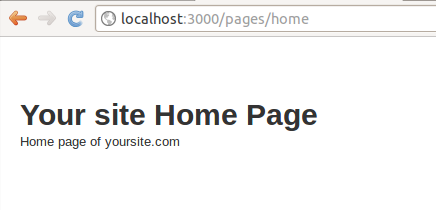
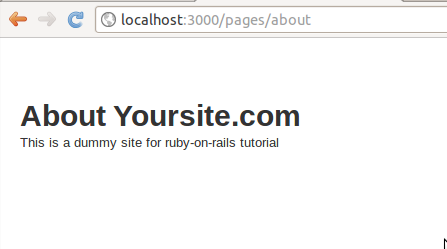


Figure: Screenshot of /pages/home

Figure: Screenshot of /pages/about

Test Output



Click 'Run test' button above to verify if your task solution is right.

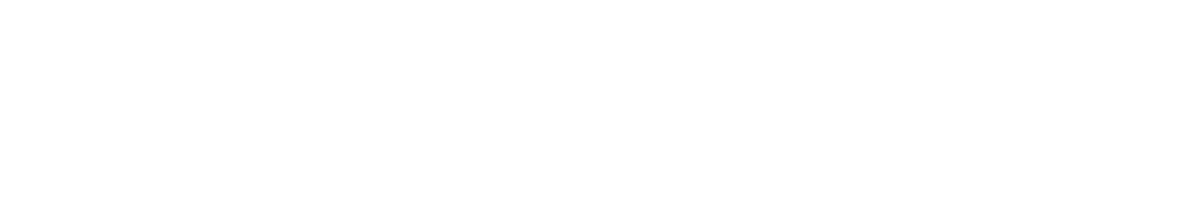
[← Previous](#_bookmark11) [Next →](#_bookmark16)

© Codelearn 2012

[**Follow**](#_bookmark16) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark16) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Understanding Rails model view controller (MVC)

[Module 1. Getting started with Rails](#_bookmark17)



[← Previous](#_bookmark13) [Next →](#_bookmark18)

In the last lesson - [Adding pages from Rails controller](#_bookmark16) - you created Pages controller with two actions - home and about.

|  |
| --- |
| [1. Zero to app in 5](#_bookmark0) [steps](#_bookmark0) |
| [Default app -](#_bookmark1) [your\_site](#_bookmark1) |
| [2. Rails directory](#_bookmark4) [structure](#_bookmark4) |
| [3. Say hello through](#_bookmark6) [Rails](#_bookmark6) |
| [After hello, say](#_bookmark8) ['greetings'](#_bookmark8) |
| [4. Pages using](#_bookmark11) [controller](#_bookmark11) |
| [Home & about](#_bookmark13) [pages](#_bookmark13) |
| [5. Visualizing Rails flow](#_bookmark16) |

In this lesson, you will dive deeper into the Rails controller, routes and views also referred as MVC architecture.

To understand what happens when a user types a URL, the following sequence is followed

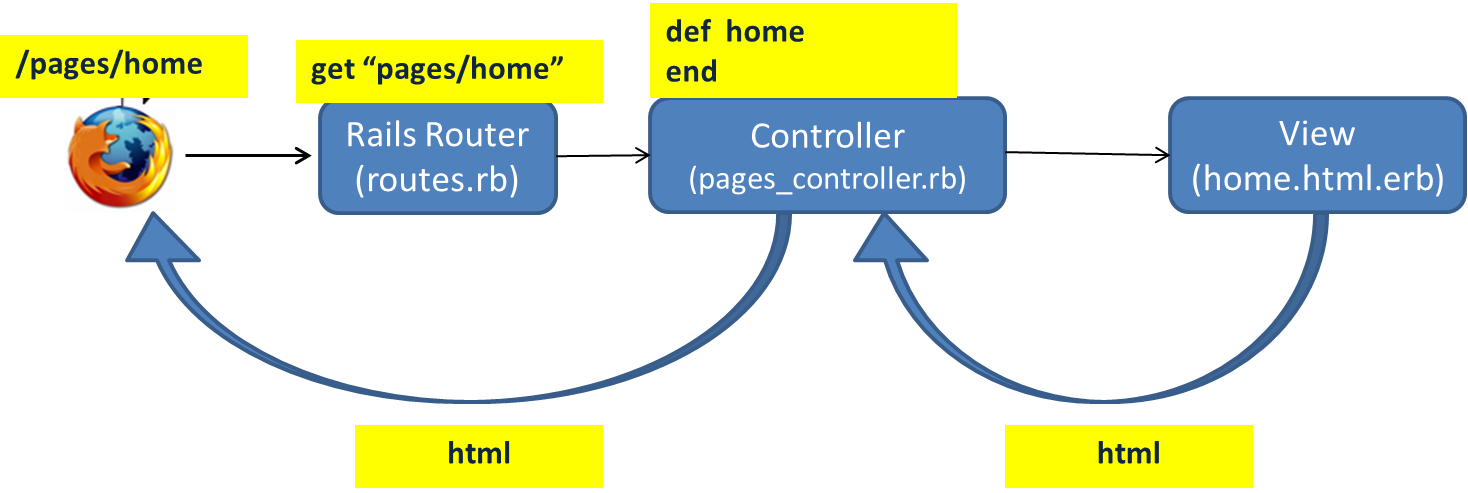


Figure: Interactions between Router --> Controller --> View

[Module 2. Starting with to-do list app](#_bookmark16)

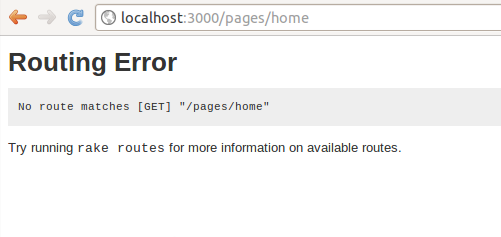
[Module 3. Add, delete, mark &](#_bookmark16) [unmark todos](#_bookmark16)

1. User types URL http://localhost:3000/pages/home , assuming the rails server is running , the request first reaches the Rails router.
2. The Router checks the config/router.rb file to see if there is an entry matching the URL requested. In our case there is a route entry - get "pages/home", so the request is forwarded to the Pages controller, home action.

If there are no matching routes, a routing error is shown to the user as below.

Figure: Router Error.

3. The request then reaches Pages controller, home action. The method is empty.



1. Each action fetches the view file associated with it, populate it with data (if any) & sends it to the browser. home action fetches app/views/pages/home.html.erb file and show the final HTML in the browser.
2. Note In this particular case, there is no interaction of controller with model.

Only routes, controller & view are involved in the complete transaction.

A quick recap :-

* 1. Whenever a URL is entered into the browser, the router checks whether there is a corresponding entry for the request in the config/routes.rb
  2. If there is a route match, the request goes to the appropriate action of the controller.

3. The action fetches the corresponding view & shows it on the browser.

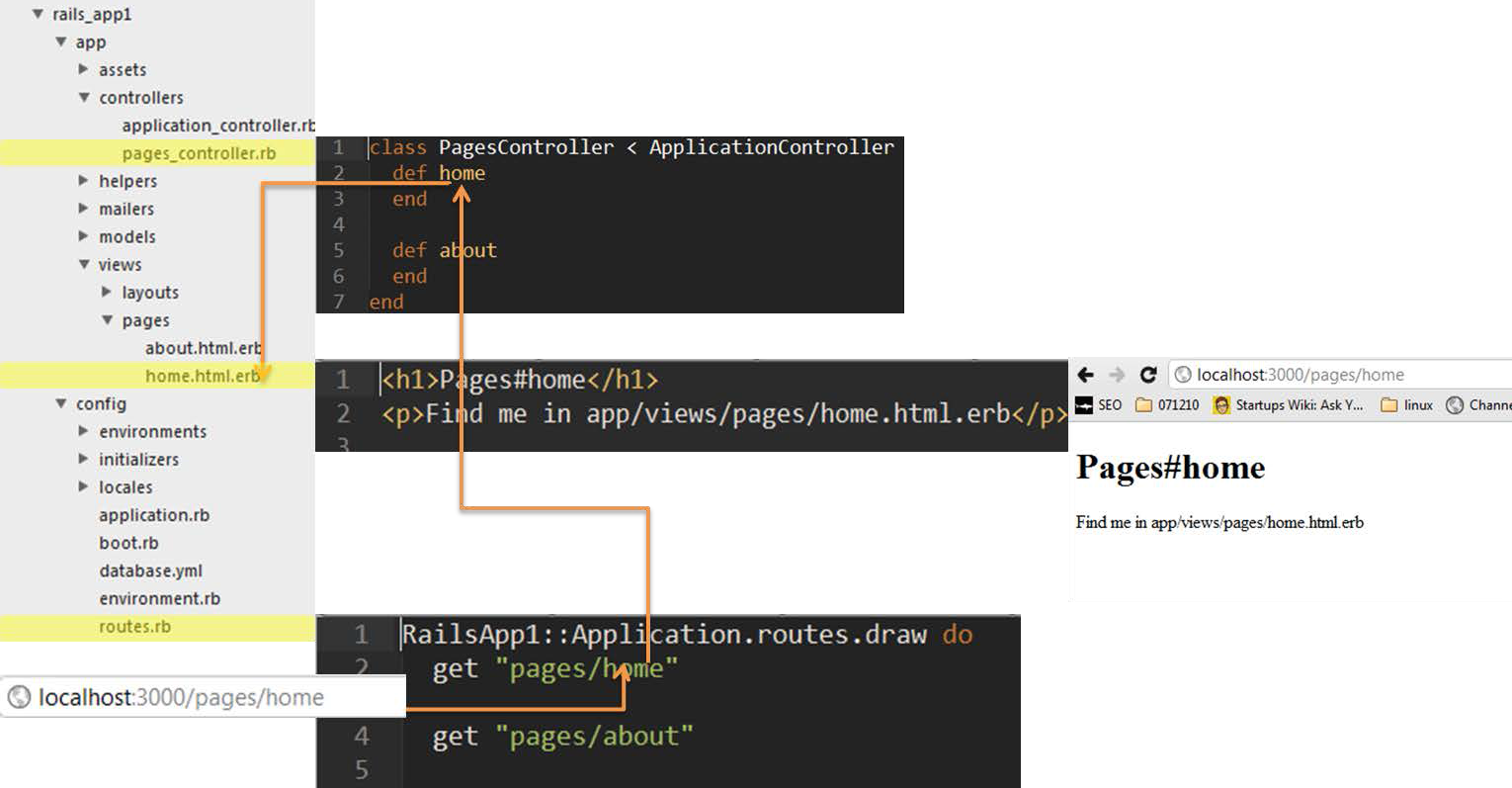


Figure: Alternate view of Rails control flow

##### Adding a blog page in 4 easy steps.

In the last lesson we understood more about controllers. In this lesson, we are going to deepen our understanding about controllers, and views by creating another URL and page.

http://localhost:3000/pages/blog

1. add a route in

config/routes.rb

* 1. get 'pages/blog'

2.

1. Add an empty action to pages controller

app/controller/pagesController.rb

* 1. def blog
  2. end

1. Create the default view, and populate it with content

app/views/pages/blog.html.erb

* 1. <title>Blog title</title>
  2. <h1>Pages Blog </h1>

4. Now view localhost:3000/pages/blog in the output pane

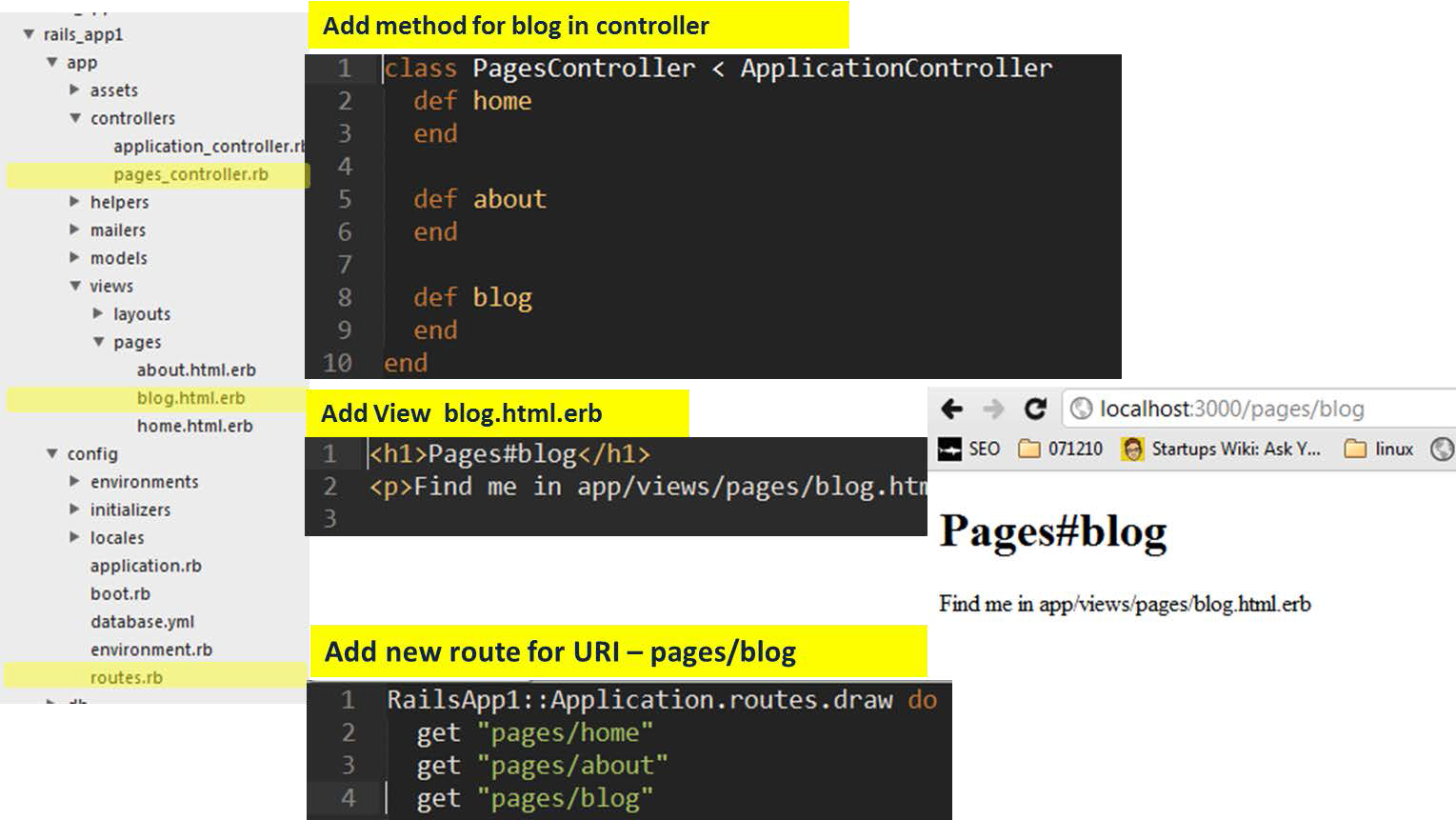


Figure: Adding a new page - blog

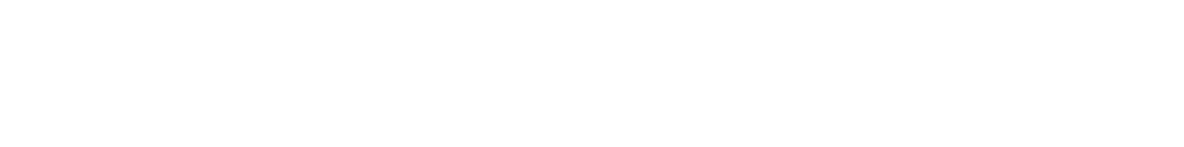
[← Previous](#_bookmark13) [Next →](#_bookmark18)

© Codelearn 2012

[**Follow**](#_bookmark18) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark18) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - How to design a Rails app

[Module 1. Getting started with Rails](#_bookmark18)



[← Previous](#_bookmark16) [Next →](#_bookmark61)



[Module 2. Starting with to-do list app](#_bookmark19)

The previous Module 'Learning Rails basics' was a quick run through a Rails app by letting you create a quick website - your\_site. You created an app, fiddled with Gemfile, added a hello world page & learnt to create pages using Rails controllers.

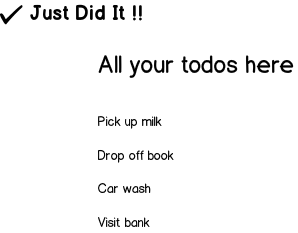
|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

Usually a web application is lot more complex then just a simple static website that you built in previous module.

For the rest of the course, we decided to pick a to-do list app. A to-do list app lets you log things that you want to do, edit them & mark them complete once you are done with them.

The following screenshots depicts the stage by stage development of the to- do list app in this module. The first image is the mockup of the app. Rest of the images is how the app would look at the end of respective lessons.

[Mockup (Lesson 1)](#_bookmark20) [Lesson 2 Lesson 3 Lesson 4](#_bookmark18)



[Module 3. Add, delete, mark &](#_bookmark18) [unmark todos](#_bookmark18)

Figure : Mockup of the to-do list app.

In [Module 1 - lesson 5](http://www.codelearn.org/tutorial/1/Module-Name/6/Lesson-Name), the Rails basics like controllers, views, routes etc were introduced.

In this lesson, we will map elements of to-do list app to the Rails flow.

Note Before writing a single line of code, experienced web developers follow the steps below to design the app to limit its scope.

### Q1. What the app should do ?

###### Listing requirements

The basic to-do list app should have the following features :

1. Adding todos
2. Listing todos
3. Marking them complete
4. Unmarking them, if they have been accidentally marked as complete
5. Deleting any extra todos (or todos you are no longer interested to complete)

6. Extracting todos from your mind & adding to the list (just kidding!)

### Q2. How does the browser interact with the app ?

###### Deciding on the urls based on the requirements

Take a requirement - adding todo. The url for adding todos can be

http://localhost:3000/todos/add

Note The url could also be /todos\_add instead of /todos/add. It is totally on the rails developer to decide the structure of the urls. Rails routes lets the developers change the url structure on the go without affecting the rest of the architecture.

Similarly the other urls can be :-

1. /todos/index
2. /todos/mark
3. /todos/unmark
4. /todos/delete

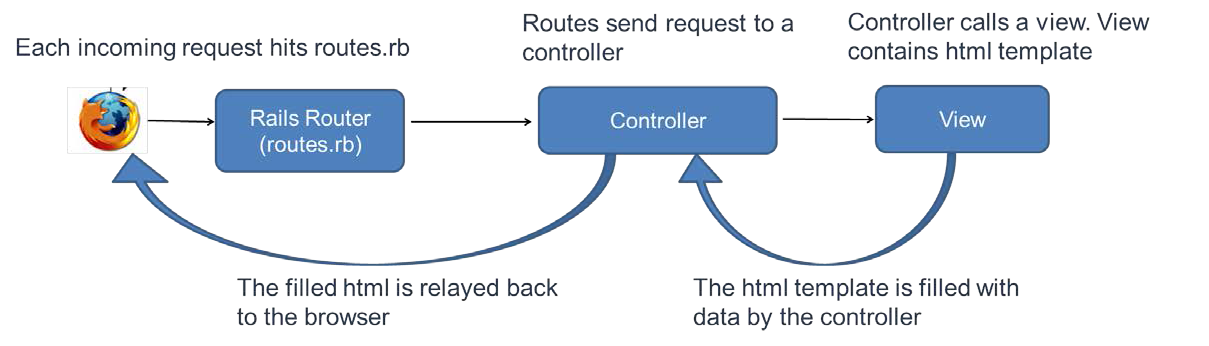
### Q3. How do the urls map to rails flow ?

###### Generating controllers based on our requirement

Before we go ahead & map the url's, lets revisit the rails flow once again.

Figure: Revisiting rails flow before mapping.

Now that we have decided on the URLs, we need to design the routing & appropriate controller to handle these URLs.

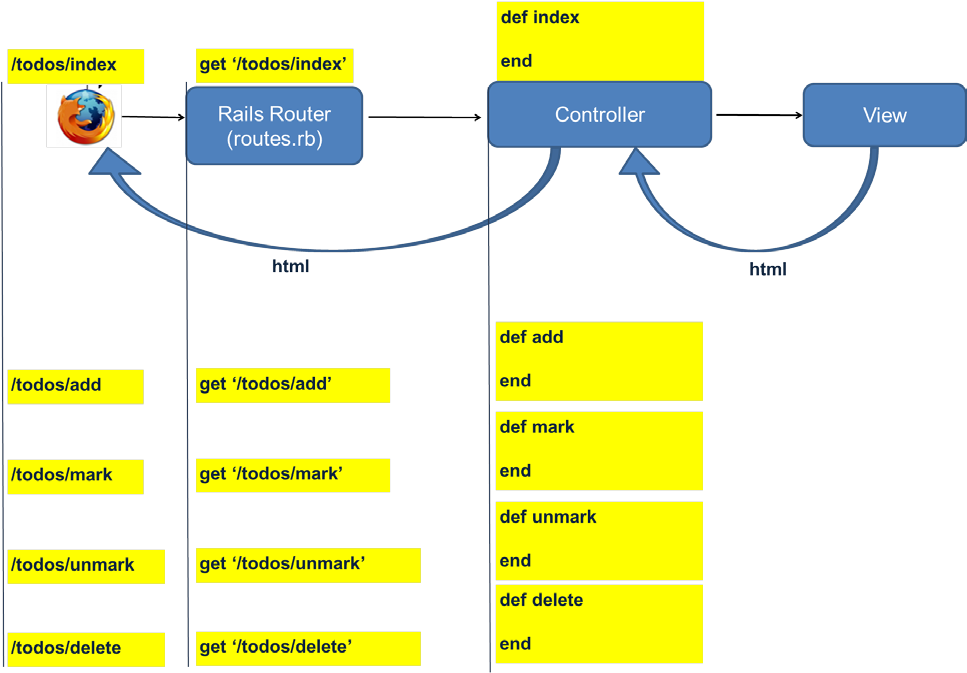


Important Each url is handled by an action of a controller. The action is a function that handles URLs, interacts with the database and renders an HTML page (also called view) appropriately.

For the current app, the URLs when mapped to their appropriate actions would look something like this.

Figure: Mapping URLs to rails.

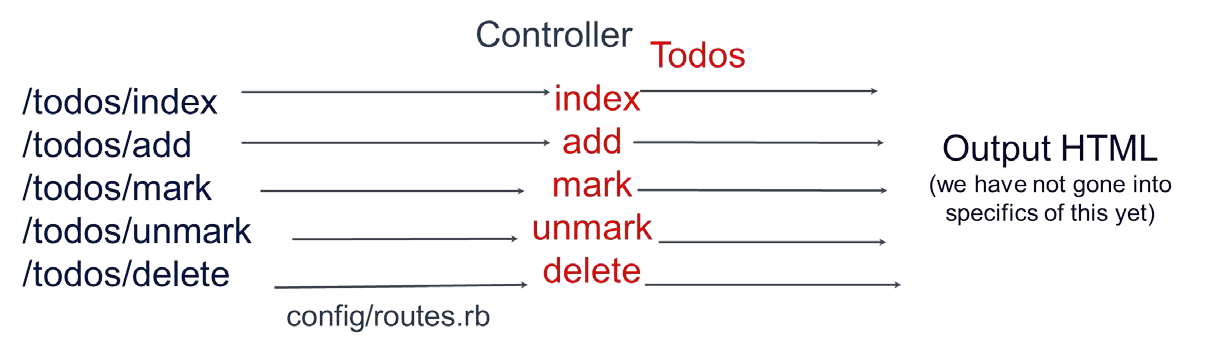
The urls are handled by the actions index , add, mark, unmark & delete of the controller Todos.



Hurray We have successfully designed the controller structure for our app.

Figure: To-do list app with controllers & actions.

### Q4. How many views are needed for the app ?



###### Optimizing views

Rails creates one view for every action that is created. But do we really need all the views ? Below is the list of files generated by Rails for the todo app.

index.html.erb add.html.erb mark.html.erb unmark.html.erb delete.html.erb

A bit of introspection would reveal that after every action, we would need to list all the todos. For eg. - if a new todo is added, control needs to be passed to the index action which would in-turn display all the todos.

Also, no other action apart from the index action needs a view. They simply interact with the database (by adding a todo or marking todo complete) & then hand the control over to the index action.

Only index action needs a view file - index.html.erb. Rest of the view files should be removed.

Also, other actions should redirect to index action after doing their respective jobs.

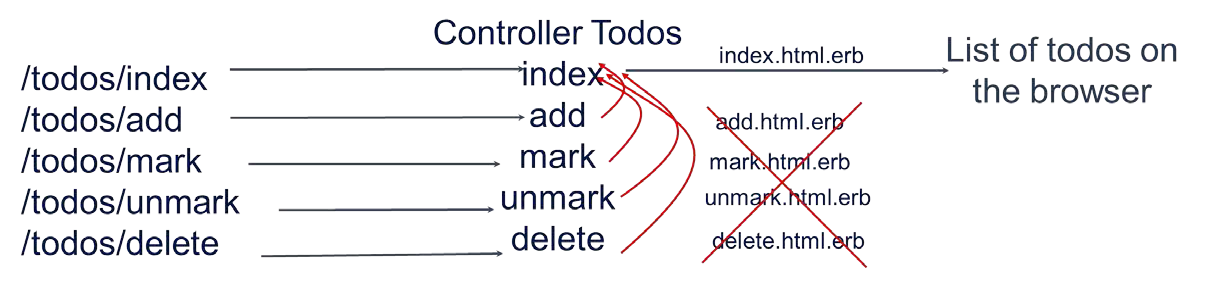
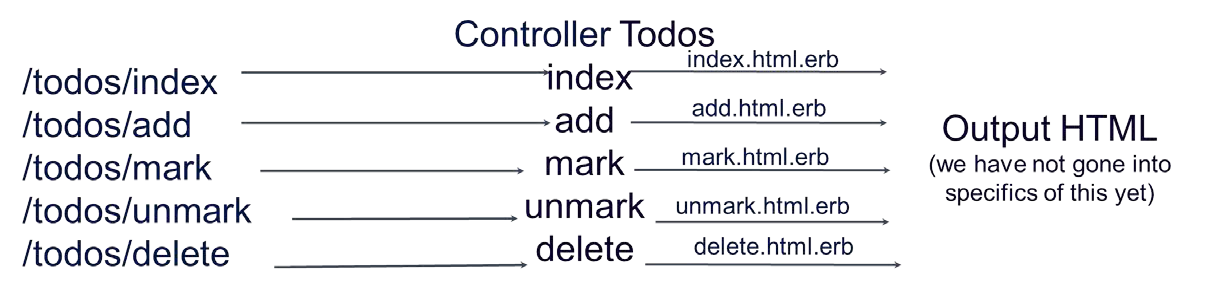


Figure: App before view optimization.

Figure: App after view optimization.

### Q5. How data is handled by the rails app ?

###### Designing models

Our app need to handle the data persistently. If a todo is added & the next time you visit the app, it should be able to show you all the todos that have been added so far.

Important In Rails, models provide an interface to store & query the database.

The model name is Todo

Note Controllers & models can have same name but they are different.

Our Todo model has two fields - name & status

Another note We would add more requirements later, but this is the absolute bare minimum requirement).

The name could be a string like 'Buy Milk' & the status could be boolean - 0 for pending & 1 for complete.

**Hurray** We have successfully designed the Todo model for the app

To conclude, lets have a look at our to-do list app control & data flow.

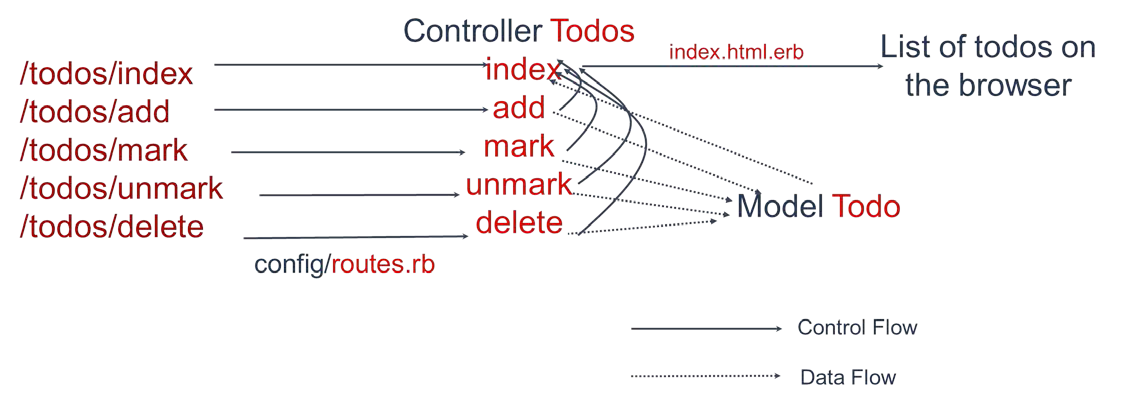


Figure: To-do list app final design.

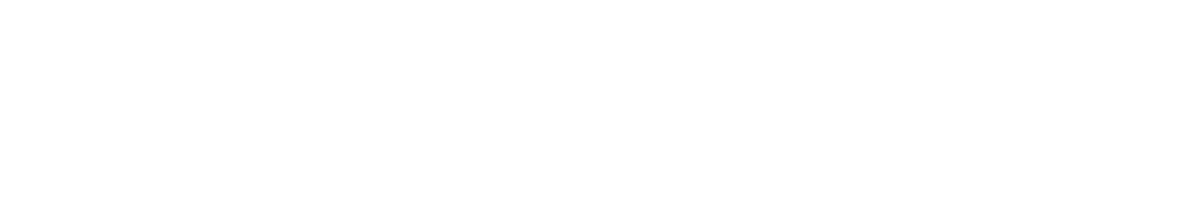
[← Previous](#_bookmark16) [Next →](#_bookmark61)

© Codelearn 2012

[**Follow**](#_bookmark21) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark21) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Include modal, modify delete to use link\_to & redirect\_to

[Module 1. Getting started with Rails](#_bookmark21)



[← Previous](#_bookmark38) [Next →](#_bookmark41)

[Module 2. Starting with to-do list app](#_bookmark22)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In the '[Delete last todo](#_bookmark35)' task, you added a link to delete the last todo. This task require to change the link to use link\_to, include Twitter Bootstrap modal window asking for deletion confirmation & redirecting the user to

/todos/index post deletion.

1. Start with the app created in the last task - 'Bootstrap CSS with navbar'. You should have an app with Twitter Bootstrap css with navigation bar.
2. Follow later part of the parent lesson 'Add Twitter Bootstrap CSS' to add a named route for delete action & replace 'a href' link with link\_to

config/routes.rb

~~get "todos/delete"~~

match "todos/delete" => "todos#delete", :as => 'delete'

app/views/todos/index.html.erb

~~<a href="/todos/delete">Delete last todo</a>~~

<%= link\_to "Delete last todo", delete\_path %>

Test after this point that 'Delete last todo' link is working in the app.

1. Modify link\_to to look like red button as in the app screenshot below.

app/views/todos/index.html.erb

<%= link\_to "Delete last todo", delete\_path, :class => '*red button class name*'

%>

[Module 3. Add, delete, mark &](#_bookmark21) [unmark todos](#_bookmark21)

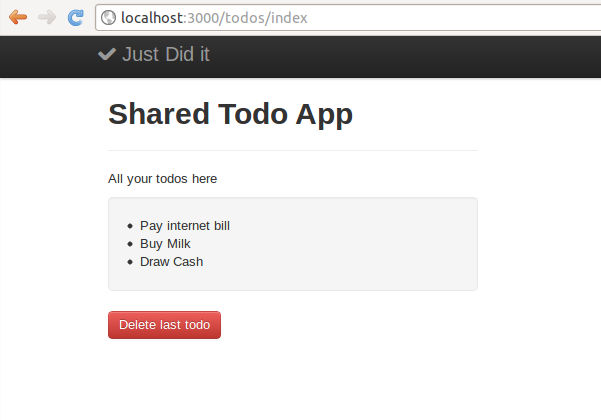
Figure out the right class name to be inserted from

<http://twitter.github.com/bootstrap/base-css.html#buttons>

You should have 'Delete last todo' showing as red button in the app. The link should work fine.

Figure: Todo app with bootstrap.css, with delete todo button

1. Add Bootstrap modal. Modals are like popups (look at the screenshot below). Read more about modals here - <http://twitter.github.com/bootstrap/javascript.html#modals>



Instead of control going to '/todos/delete', the modal popup should come up on clicking the link 'Delete last todo'.

For the modal window to work, copy the bootstrap.js which is part of the zip file you downloaded in the parent lesson.

Note It is assumed that you are inside shared\_todo\_app in the console before you execute the command below.

Console

user\_1@codelearn:~/shared\_todo\_app$ cp ~/bootstrap/js/bootstrap.js app/assets/ja vascripts/bootstrap.js

The documentation of the modal popup suggests that a few extra

attributes need to be added to the link. Change the link\_to as shown below

app/views/todos/index.html.erb

<%= link\_to "Delete last todo", delete\_path"#*id of the modal window*", "data-to ggle" =>"modal" , :class => ' *red button class name*' %>

Add the modal code anywhere in index.html.erb. The code is picked from the Twitter Bootstrap site. The content in italics need to be filled by you.

app/views/todos/index.html.erb

<div class="modal hide fade" id="*id of the modal window*">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal" aria-hidden="true"

>×</button>

<h3>Modal header</h3>

</div>

<div class="modal-body">

<p>One fine body…</p>

</div>

<div class="modal-footer">

<a href="#" class="btn" data-dismiss="modal" >cancel</a>

<a href="#" class="btn btn-primary">Save changes</a>

</div>

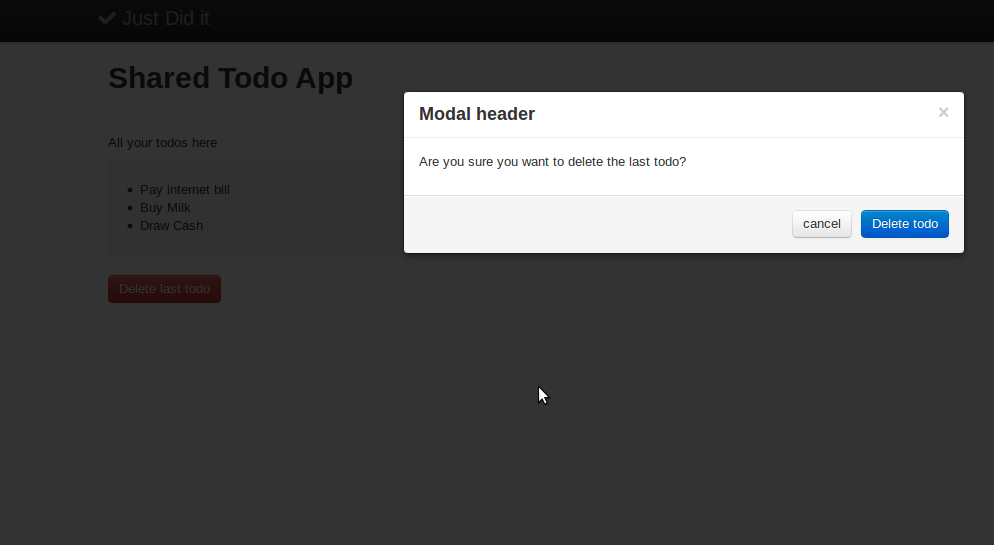
</div>

The link\_to is connected to the modal window via the 'id' attribute. If the modal window has id="modalid" <div class="modal hide fade" id="modalid"> , the link\_to should have the href field as "#modalid" <% link\_to "Delete last todo", "#modalid" %> . Note the '#' in the link\_to href

At this point, you should have a modal window which is normally hidden & shows up when 'Delete last todo' link is clicked. Clicking on 'x' or 'cancel' button should hide the modal window.

Figure: Todo app with bootstrap.css, with delete confirmation modal

1. The Bootstrap modal is not usable yet. You need to change the 'Save changes' button to 'Delete todo'. Do the changes that are highlighted.



app/views/todos/index.html.erb

<div class="modal hide fade" id="*id of the modal window*">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal" aria-hidden="true"

>×</button>

<h3>Modal header</h3>

</div>

<div class="modal-body">

<p>One fine body…*add the body as shown in the screenshot*</p>

</div>

<div class="modal-footer">

<a href="#" class="btn" data-dismiss="modal" >cancel</a>

~~<a href="#" class="btn btn-primary">Save changes</a>~~

*add the link\_to code which is blue in color & has value 'Delete todo' a s shown in the screenshot below*

</div>

</div>

At this point, the 'Delete todo' button inside modal window should delete last todo by sending control to /todos/delete.

1. Post deletion the user goes to the url /todos/delete currently. As per first lesson, [Designing to-do list app](#_bookmark18), the control should be redirected to

/todos/index post deletion. You need to use 'redirect\_to' inside the delete action for it.

Note 'redirect\_to' is introduced at the end of this task

app/controllers/todos\_controllers.rb

def index

@todo\_items = Todo.all redirect\_to :index

end

The app now is ready to pass the tests.

The visual behavior of the app at the end of the task should be as follows :-

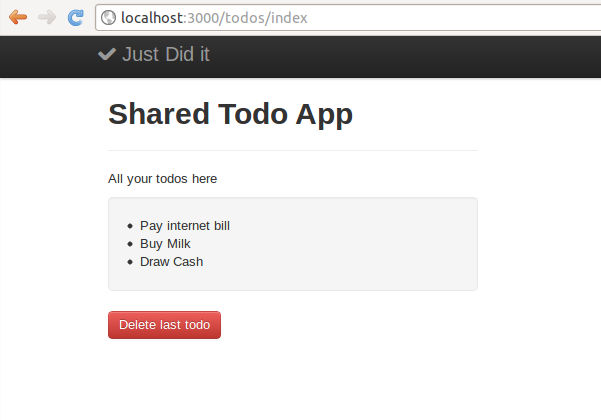
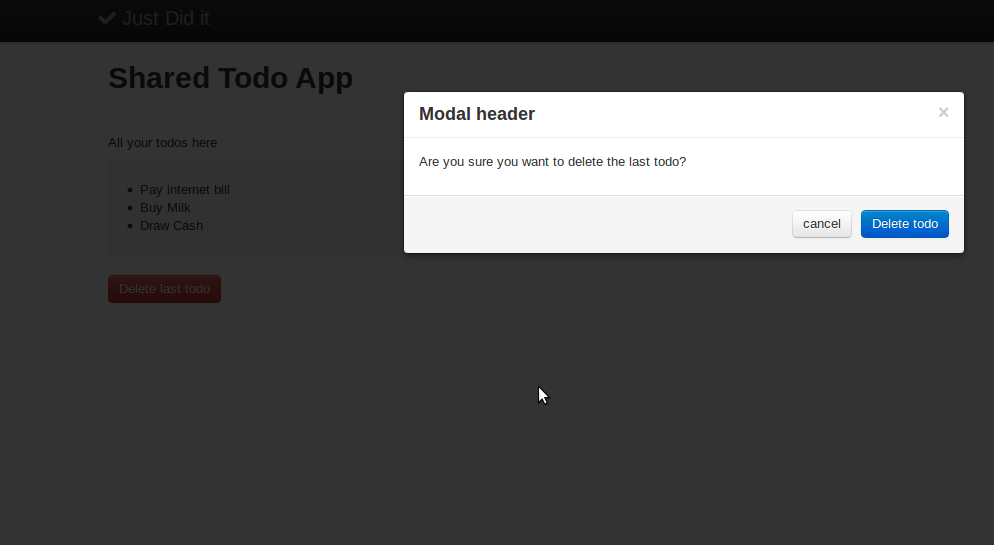


Figure: Todo app with bootstrap.css, with delete todo button

After the delete button is clicked, an alert should appear asking for delete confirmation. (again refer below diagram)

Figure: Todo app with bootstrap.css, with delete confirmation modal

##### Introducing redirect\_to



There is a controller function redirect\_to which is in similar league as render . redirect\_to :index should redirect the control to index action post deletion. You can safely remove app/views/todos/delete.html.erb as it is not needed anymore.

If you give a name to the route for index action as, lets say, index; then you can also do redirect\_to index\_path .

Note that render can only render a view, while redirect\_to redirects the control to an action of same or even different controller. Hence render index\_path does NOT make sense, but redirect\_to index\_path does.

Also :index in render :index refers to app/views/todos/index.html while :index

in redirect\_to :index refers to index action of Todos controller.

Test Output

Click 'Run test' button above to verify if your task solution is right.

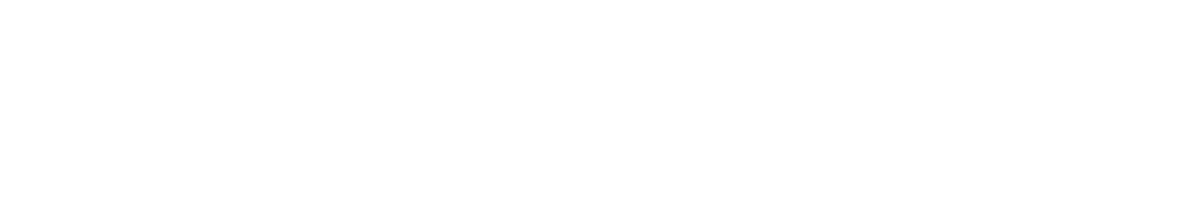
[← Previous](#_bookmark38) [Next →](#_bookmark41)

© Codelearn 2012

[**Follow**](#_bookmark24) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark24) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Display one hardcoded todo from controller

[Module 1. Getting started with Rails](#_bookmark24)



[← Previous](#_bookmark61) [Next →](#_bookmark27)

[Module 2. Starting with to-do list app](#_bookmark25)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

This task makes you do all things we taught in the parent lesson - [Todos from](#_bookmark61) [controller](#_bookmark61).

Essentially follow the exact same steps in this lesson, and display one todo within an <li> tag & not within a '<p>'.

app/views/todos/index.html

<ul>

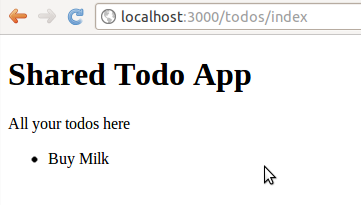
<li> #todo item here </li>

</ul>

Once done, /todos/index in the App Output should show the app as below.

Figure: Todo app screenshot after Task1 is completed

[Module 3. Add, delete, mark &](#_bookmark24) [unmark todos](#_bookmark24)



Test Output

Click 'Run test' button above to verify if your task solution is right.

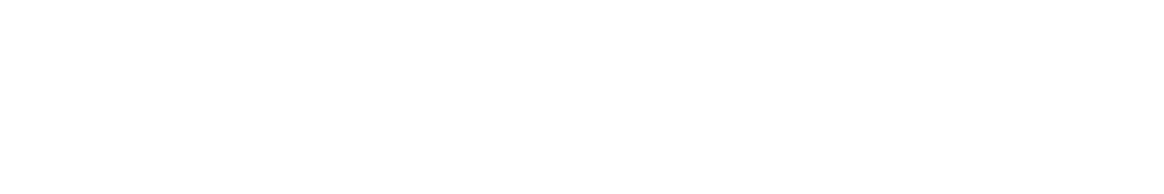
[← Previous](#_bookmark61) [Next →](#_bookmark27)

© Codelearn 2012

[**Follow**](#_bookmark27) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark27) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

## Rails tutorial - Task - Display multiple hardcoded todos from controller using array

[Module 1. Getting started with Rails](#_bookmark27)



[← Previous](#_bookmark24) [Next →](#_bookmark30)



[Module 2. Starting with to-do list app](#_bookmark28)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In the last task - [Display one todo](#_bookmark24), you displayed 1 todo. In this task you will have to display todos from an array.

1. Start with the Rails app created in the last task - [Display one todo](#_bookmark24).
2. Remove @todo\_item1 instance variable in app/controllers/todos\_controllers.rb. Replace it with @todo\_array as shown below.

@todo\_array = [ "Buy Milk", "Buy Soap", "Pay bill", "Draw Money" ]

1. Edit app/views/todos/index.html to show all the todo items wrapped inside "<li>" tag in the view.

app/views/todos/index.html

<ul>

<% @todo\_array.each do |t| %>

<li> #todo item here </li>

<% end %>

</ul>

Hint Follow the Ruby tutorial in the parent lesson - [Todos from controller](#_bookmark61), to check how arrays are parsed using 'each'.

1. Once done, /todos/index in App Output should show as below.

[Module 3. Add, delete, mark &](#_bookmark27) [unmark todos](#_bookmark27)

Figure: Todo app screenshot after Task 2 is completed

Test Output



Click 'Run test' button above to verify if your task solution is right.

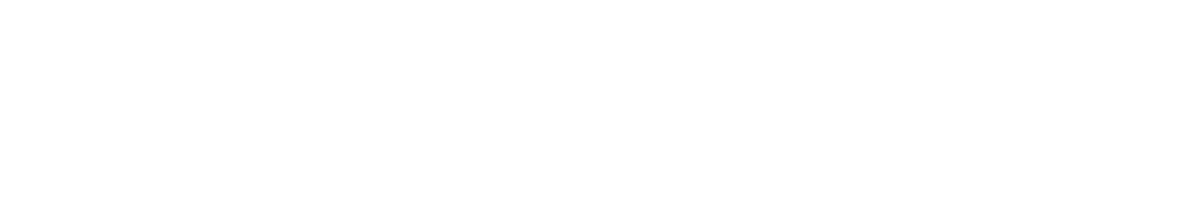
[← Previous](#_bookmark24) [Next →](#_bookmark30)

© Codelearn 2012

[**Follow**](#_bookmark30) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark30) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Rails models create, migrate, rollback and destroy

[Module 1. Getting started with Rails](#_bookmark30)



[← Previous](#_bookmark27) [Next →](#_bookmark32)

[Module 2. Starting with to-do list app](#_bookmark31)

In the last lesson - [Todos from controller](#_bookmark61), you learnt to use controllers to show data on the view. But the data was essentially static.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

A real world app needs dynamic content. You need to be able to store the dynamic content, be able to retrieve it and display it. Typically a database is used to store the dynamic data.

In this lesson, you will learn to use models to store todos in database through Rails Console & then fetching the same via controller to display it on the view.

### Generating Todo model

Rails interact with the database through models. According to design as per

[Designing to-do list app](#_bookmark18), our model is named Todo.

Lets start with just one field todo\_item of type string for the model Todo. The todo\_item holds the name of the todo like 'Buying Milk'.

The way to generate a model in rails is by using rails generate model command :-

[Module 3. Add, delete, mark &](#_bookmark30) [unmark todos](#_bookmark30)

Console

user\_1@codelearn:~/shared\_todo\_app$ rails generate model Todo todo\_item:string invoke active\_record

create db/migrate/20120802113330\_create\_todos.rb create app/models/todo.rb

invoke test\_unit

create test/unit/todo\_test.rb create test/fixtures/todos.yml

Convention When you create a model, the name is singular and the name starts

with capital letter. Todo, instead of Todos.

If you end up doing a mistake you can always undo the creation of a model.

Console

user\_1@codelearn:~/shared\_todo\_app$ rails destroy Todo invoke active\_record

remove db/migrate/20120802113330\_create\_todos.rb remove app/models/todo.rb

invoke test\_unit

remove test/unit/todo\_test.rb remove test/fixtures/todos.yml

The destroy command removes the files it has created previously, leaving no trace whatsoever.

### Introducing Migration

The 'rails generate model' command also creates a migration file

db/migrate/20120802113330\_create\_todos.rb.

Note The exact name of the file would be different for you. Look at the rails generate model command output to get the right name

db/migrate/20120802113330\_create\_todos.rb

* 1. class CreateTodos < ActiveRecord::Migration
  2. def change
  3. create\_table :todos do |t|

4. t.string :todo\_item

5.

1. t.timestamps
2. end

8. end

9. end

The migration file is like schema which defines the database table structure. The table name is todos with two fields todo\_item and timestamps . We will understand what timestamps does in a while.

rake db:migrate pushes the database changes from the migration file to the actual database. In Rails the default database is sqlite.

Console

user\_1@codelearn:~/shared\_todo\_app$ rake db:migrate

== CreateTodos: migrating ====================================================

-- create\_table(:todos)

-> 0.0022s

== CreateTodos: migrated (0.0024s) ===========================================

One of the purpose of Rails model is to shield us from dealing directly with sql database. Writing sql queries for every database task is tedious and rails

handles all that for us.

Tip You would hardly need to refer to the actual table by the name todos. You would only be accessing the data through the model Todo.

How to undo a model creation? You can use rails destroy to destroy the model. However if you have executed rake db:migrate, you need to execute rake db:rollback as shown below.

Console

user\_1@codelearn:~/shared\_todo\_app$ rake db:rollback

This will remove the table and all the data from your database as well.

### Fidlling with Models inside Rails console

You used rails console in last lesson to learn a little Ruby. In this lesson, we will use it to fiddle with the database tables. Type rails console in the Codelearn Console, followed by enter key to enter Rails console.

Console

user\_1@codelearn:~/shared\_todo\_app$ rails console

##### Todo structure

Entering the model name Todo will display the structure of the database table

Rails console

1.9.3p194 :002 > Todo

=> Todo(id: integer, todo\_item: string, created\_at: datetime, updated\_at: datetime

)

As you see , it has the fields

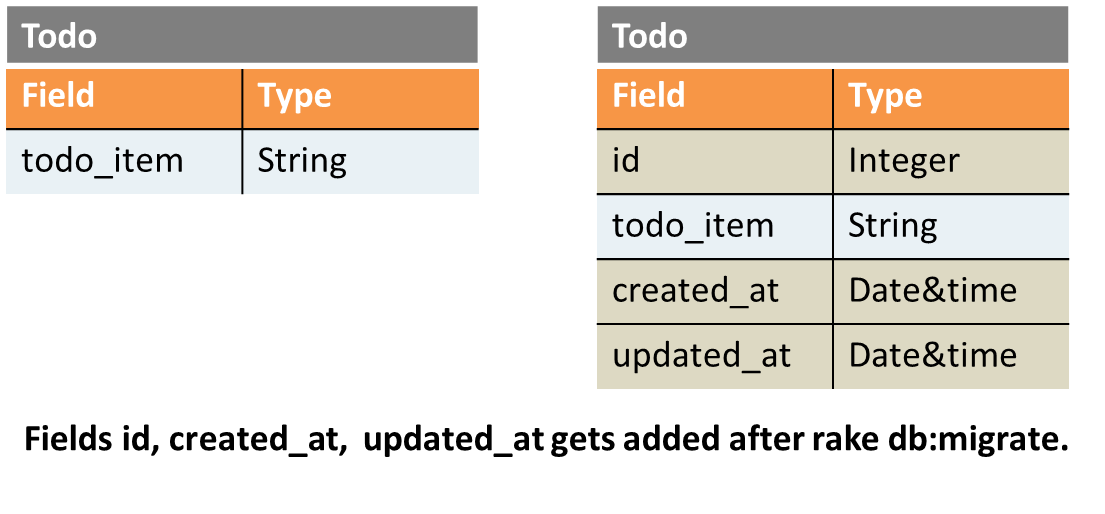
id - Automatically added by rails. Every new table row gets a new id (autoincrement). All tables have id.

todo\_item, type string, holds actual todos, entered by us when creating the Todo model.

created\_at & updated\_at both of type datetime, created by rails for all tables. created\_at stores time of creation of table, updated\_at stores time when table was last updated.

Figure: When the model Todo gets added to db by rake:db migrate, additional fields get added.

##### Fetching all todos



Todo.all displays all the rows of the database table. In this case, the result is empty string [].

1.9.3p194 :004 > Todo.all

Todo Load (0.3ms) SELECT "todos".\* FROM "todos"

=> []

##### Creating new todos

Follow the steps below to create a new todo

1. Use t = Todo.new to create an empty row in the Todo table and stores it in t

Rails console

1.9.3p194 :001 >t = Todo.new

=> #

1. t.todo\_item = "pick up milk" populates values into the fields of the row.

Rails console

1.9.3p194 :002 > t.todo\_item = "pick up milk"

=> "pick up milk"

1. t.save saves the row into the database table

Rails console

1.9.3p194 :003 > t.save (0.1ms) begin transaction

SQL (33.3ms) INSERT INTO "todos" ("created\_at", "todo\_item", "updated\_at") VA LUES (?, ?, ?) [["created\_at", Sat, 18 Aug 2012 09:28:24 UTC +00:00], ["todo\_i

tem", "pick up milk"], ["updated\_at", Sat, 18 Aug 2012 09:28:24 UTC +00:00]] (89.0ms) commit transaction

=> true

When you do t.save, the row gets a id and the created\_at and updated\_at also get populated. This is when the row actually gets stored into the database.

Notice the line "commit transaction", means row is getting stored to database.

At this stage created\_at and updated\_at have the same value.

Todo.all displays all rows in the table, in this case just one row. The rails converts Todo.all into SELECT "todos".\* FROM "todos"

Rails console

1.9.3p194 :004 > Todo.all

Todo Load (0.2ms) SELECT "todos".\* FROM "todos"

=> [#<Todo id: 5, todo\_item: "pick up milk", created\_at: "2012-08-18 09:28:24", u pdated\_at: "2012-08-18 09:28:24">]

##### Creating todo using create function

You can also create a row using Todo.create, no need of t.save in this case. Notice "commit transaction" line again here.

Rails console

1.9.3p194 :005 > Todo.create(:todo\_item => "Pay internet bill") (0.1ms) begin transaction

SQL (0.6ms) INSERT INTO "todos" ("created\_at", "todo\_item", "updated\_at") VALUES (?, ?, ?) [["created\_at", Sat, 18 Aug 2012 09:55:33 UTC +00:00], ["todo\_item", "P

ay internet bill"], ["updated\_at", Sat, 18 Aug 2012 09:55:33 UTC +00:00]] (339.1ms) commit transaction

=> #<Todo id: 6, todo\_item: "Pay internet bill", created\_at: "2012-08-18 09:55:33"

, updated\_at: "2012-08-18 09:55:33">

##### Selectively fetching todos

You can find the first row in database using Todo.first, last row Todo.last You can find a row with a particular id , Todo.find(1) - row with id = 1,

Todo.find(2) - row with id = 2, and so on

Searching for a row with a field content is also possible - Todo.find\_by\_field\_name("field\_value") . Eg - Todo.find\_by\_todo\_item("pick up milk").

Rails console

1.9.3p194 :001 > Todo.first

Todo Load (0.2ms) SELECT "todos".\* FROM "todos" LIMIT 1

=> #

1.9.3p194 :003 > Todo.find(5)

Todo Load (0.2ms) SELECT "todos".\* FROM "todos" WHERE "todos"."id" = ? LIMIT 1 [["id", 5]]

=> #

1.9.3p194 :008 > Todo.find\_by\_todo\_item("pick up milk")

Todo Load (0.2ms) SELECT "todos".\* FROM "todos" WHERE "todos"."todo\_item" = 'pic k up milk' LIMIT 1

=> #

##### Deleting todos

Deleting a row in database is fairly easy t = todo.first, t.delete You can delete all rows of a table using Todo.delete\_all

Rails console

1.9.3p194 :009 > t= Todo.first

Todo Load (0.2ms) SELECT "todos".\* FROM "todos" LIMIT 1

=> #

1.9.3p194 :010 > t.delete

SQL (157.8ms) DELETE FROM "todos" WHERE "todos"."id" = 5

=> #

1.9.3p194 :011 > Todo.delete\_all SQL (139.3ms) DELETE FROM "todos"

=> 1

1.9.3p194 :012 > Todo.all

Todo Load (0.3ms) SELECT "todos".\* FROM "todos"

=> []

1.9.3p194 :013 >

##### Exiting from the Rails console

Once done you can exit from the console by typing 'exit' followed by enter.

Rails console

* + 1. p194 :009 > exit

user\_1@codelearn:~/shared\_todo\_app$

You will be back to linux terminal now.

### Fetching todos from database & showing it on the view

To display all todos in the database, we need to fetch all todos, pass that data to the view and have the view display it.

The expression @todo\_items = Todo.all fetches all todos from the Todo model and creates an instance variable @todo\_items which is an array.

app/controllers/todos\_controller.rb

* + - 1. class TodosController < ApplicationController
      2. def index
      3. @todo\_items = Todo.all

1. end
2. end

The array instance variable @todo\_items is accessed in the view index.html.erb

in the following way, and all the todos are displayed using the .each construct.

app/views/todos/index.html.erb

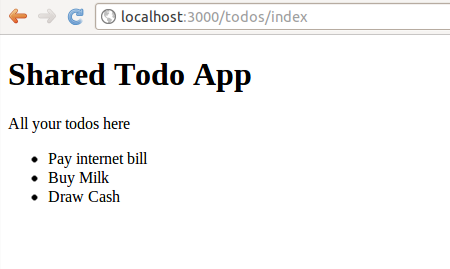
1. <title>Shared Todo App </title>
2. <h1>Shared Todo App</h1>
3. <p>All your todos here</p>

4.

1. <% @todo\_items.each do |t| %>
2. <li> <%= t.todo\_item %> </li>
3. <% end %>

Figure: Todo app with todos fetched from model

Execute above steps in the code, navigate to App Output & navigate to



/todos/index. You will see that all todos we have added to the table in the console is displayed. Try adding/deleting todos & reloading the page to see the changes.

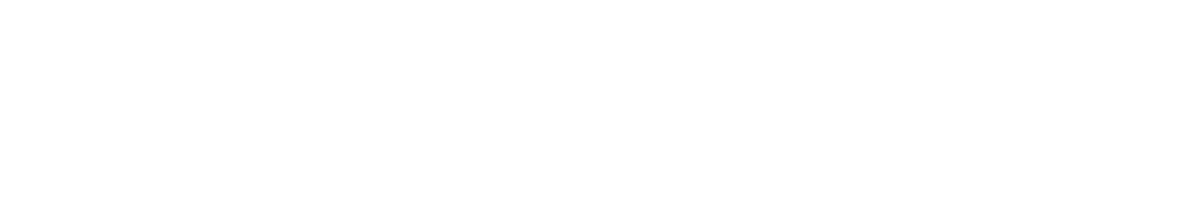
[← Previous](#_bookmark27) [Next →](#_bookmark32)

© Codelearn 2012

[**Follow**](#_bookmark32) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark32) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Store todo using Console, fetch it in controller & show on the view

[Module 1. Getting started with Rails](#_bookmark32)



[← Previous](#_bookmark30) [Next →](#_bookmark35)

[Module 2. Starting with to-do list app](#_bookmark33)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

This task you gets you upto speed, on what you learnt in the parent lesson -

[Model to store/fetch todos](#_bookmark30).

1. Start with the app created at the end of Lesson 2 - Task 2 - [Display](#_bookmark27) [multiple todos](#_bookmark27).

2. Create Todo model with a field todo\_item as told in the parent lesson.

1. Store three todos via Rails console with todo\_item names as 'Pay internet bill', 'Buy Milk' & 'Draw Cash'.
2. Edit app/controllers/todos\_controllers.rb to fetch todos from the database

& store it in an instance variable as shown in the parent lesson.

1. Modify app/views/todos/index.html.erb to show the todos from the instance variable.
2. Inside App Output in Codelearn Playground, /todos/index should look like the screenshot below.

[Module 3. Add, delete, mark &](#_bookmark32) [unmark todos](#_bookmark32)

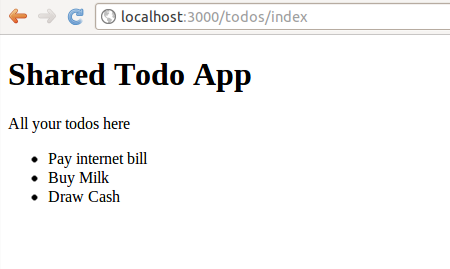


Figure: shared\_todo\_app post task completion

Test Output

Click 'Run test' button above to verify if your task solution is right.

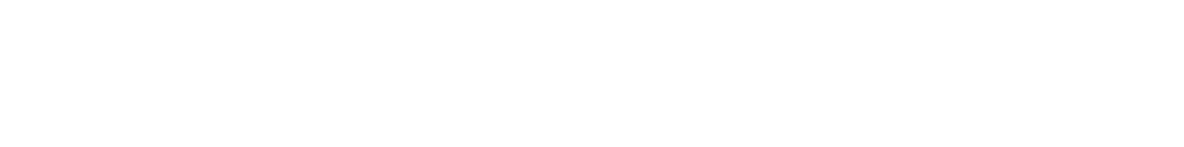
[← Previous](#_bookmark30) [Next →](#_bookmark35)

© Codelearn 2012

[**Follow**](#_bookmark35) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark35) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Implementing 'delete last todo' link

[Module 1. Getting started with Rails](#_bookmark35)



[← Previous](#_bookmark32) [Next →](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)



[Module 2. Starting with to-do list app](#_bookmark36)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In this task, we will add a 'Delete last todo' link in the view. It will require you to introduce a new URL - /todos/delete. This task will teach you how to add a new action to existing controller, add corresponding view & route entry to handle a new URL.

Add delete functionality to the app by adding a 'Delete last todo' link in the index view - /todos/index. When this link is clicked, the last row of the todos table should get deleted. Follow the steps/hints.

1. Start with the app created at the previous task - [Todos from database](#_bookmark32).
2. Add a delete action inside Todos controller

app/controllers/todos\_controller.rb.

app/controllers/todos\_controller.rb

* 1. class TodosController < ApplicationController
  2. def index
  3. @todo\_items = Todo.all
  4. end

5.

1. def delete
2. #put delete logic here
3. end
4. end
5. Create a new view - app/views/todos/delete.html.erb.

[Module 3. Add, delete, mark &](#_bookmark35) [unmark todos](#_bookmark35)

Console

user\_1@codelearn:~/shared\_todo\_app$ touch app/views/todos/delete.html.erb

This is the view which would be visible when /todos/delete is fetched on the browser. Put following content in the file.

app/views/todos/delete.html.erb

<p>Todo successfully deleted</p>

<a href="/todos/index">Back</a%>

1. Add the line below anywhere in config/routes.rb. Rails router will map

/todos/delete to the delete action that we created.

config/routes.rb

get "/todos/delete"

1. Test if the url /todos/delete is working by accessing /todos/delete in App Output. Then check if the last todo got deleted in Rails console or by accessing /todos/index.
2. If it works fine so far, introduce a link in app/views/todos/index.html at the end. <a href="/todos/delete">Delete last todo</a> .

Below is the screenshot of the the final app.

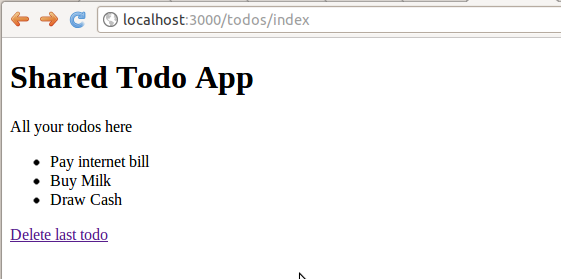


Figure: Todo app with "Delete last todo" link

Figure: Todo app with last todo deleted by clicking on the link.

Test Output

Click 'Run test' button above to verify if your task solution is right.

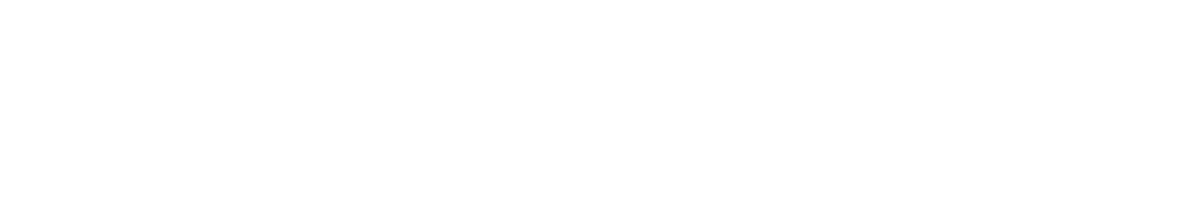
[← Previous](#_bookmark32) [Next →](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)

© Codelearn 2012

[**Follow**](#_bookmark38) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark38) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Add bootstrap CSS, insert navbar & put todos in well

[Module 1. Getting started with Rails](#_bookmark38)



[← Previous](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Next →](#_bookmark21)

[Module 2. Starting with to-do list app](#_bookmark39)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In this task, you will implement Twitter Bootstrap CSS as taught in the parent lesson - [Add Twitter Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline). You will also learn about some of the classes in Twitter Bootstrap CSS.

1. Start with the app built in the last task - [Delete last todo](#_bookmark35).
2. Follow the parent lesson 'Add Twitter Bootstrap CSS' to include the bootstrap.css in shared\_todo\_app assets directory.
3. Add top navigation bar. Get the code from [Twitter Bootstrap site](http://twitter.github.com/bootstrap/components.html#navbar). Add the code in app/views/layouts/application.html.erb. Include 'Just Did it' as <a class="brand" href="#">Just Did it</a> in the code. Remove the link code as there are no links in the app for now.

[Module 3. Add, delete, mark &](#_bookmark38) [unmark todos](#_bookmark38)

Note It does not matter if the navigation bar is not black in color. Also, the navigation bar code is added in the layout file - application.html.erb so that it is applied to all the views & not just /todos/index.

1. Modify index.html.erb by using the below code so that the todos show inside the div with class well. Keep the todos wrapped inside <li>

app/views/todos/index.html.erb

* 1. <title>Shared Todo App </title>
  2. <div class="container">
  3. <div class="row">
  4. <div class="span4">
  5. <h1>Shared Todo App</h1>
  6. <hr>
  7. <p>All your todos here</p>
  8. <div class="well">
  9. #put the code here to show the todos wrapped inside <li>
  10. </div> <!-- well -->

11. </div> <!-- span4 -->

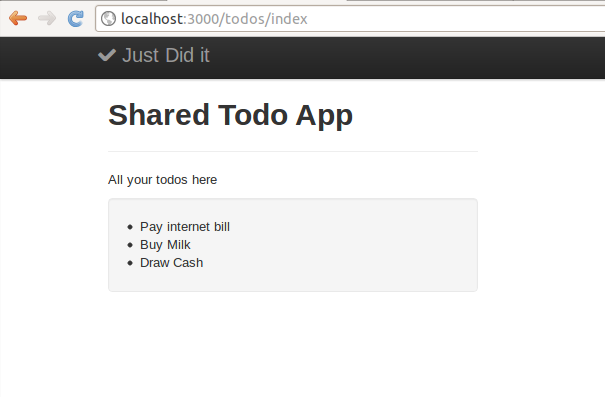
12. </div> <!-- row -->

13. </div> <!-- container -->

5. Once done, the /todos/index in App Output should look like below.

Figure: Todo app with bootstrap.css, top navbar & todos inside well.

##### Quick overview of the Twitter Bootstrap classes



In the above task, you used bootstrap classes - 'container', 'row', 'span4' and 'well'. It is good time to provide a quick intro.

Bootstrap CSS follows grid system. Grid system is the modern way of keeping HTML elements well aligned. You can read more about the grid system here [here](http://www.smashingmagazine.com/2007/04/14/designing-with-grid-based-approach/).

'container' centers the content in a 940px wide region. 'row' class signifies a row. Each Bootstrap row has 12 columns as shown [here](http://twitter.github.com/bootstrap/scaffolding.html#gridSystem). 'spanX' class spans 'X' of these columns. Eg. 'span4' covers 4 columns.

'well' class emphasizes the content by placing it inside a gray box.

To implement the navigation bar, the code has classes like 'navbar', 'navbar- inner', 'brand', 'nav' & 'active'. You do not need to worry much about them for now.

Note If you are on Google Chrome, [this resource](http://webdesign.tutsplus.com/tutorials/workflow-tutorials/faster-htmlcss-workflow-with-chrome-developer-tools/) can help you examine which class is applied to which HTML element. Additionally you can also see which CSS properties are getting applied from which class.

Test Output

Click 'Run test' button above to verify if your task solution is right.

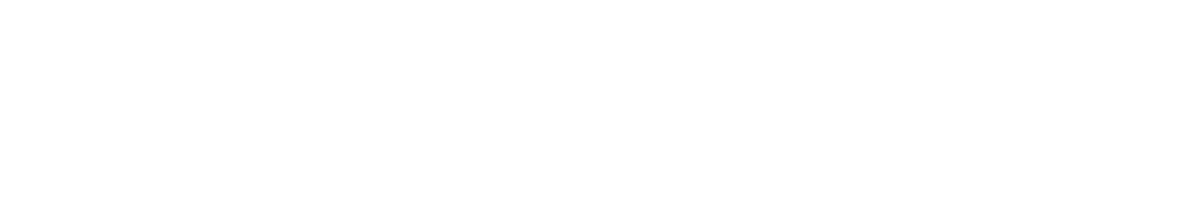
[← Previous](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Next →](#_bookmark21)

© Codelearn 2012

[**Follow**](#_bookmark41) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark41) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Introducing Ruby Gems through Twitter Bootstrap Gem

[Module 1. Getting started with Rails](#_bookmark41)



[← Previous](#_bookmark21) [Next →](#_bookmark43)

[Module 2. Starting with to-do list app](#_bookmark42)

In the last lesson '[Add Twitter Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)', we learnt how to add Bootstrap CSS by downloading & adding bootstrap.css to the Rails app.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

In this lesson, we would see an alternate easier way of doing the same thing through Bootstrap Ruby gem.

### Including Bootstrap Ruby gem

Lets get onto the action first & we will deal with how of it later. We are going to remove the traces of manually added Bootstrap files as per the last lesson. We will add the Bootstrap gem then.

1. Remove the file previously included.

Console

rm app/assets/stylesheets/bootstrap.css rm app/assets/javascripts/bootstrap.js

Check the output of the app in App Output area in Codelearn Playground. It should have become plain as below.

[Module 3. Add, delete, mark &](#_bookmark41) [unmark todos](#_bookmark41)

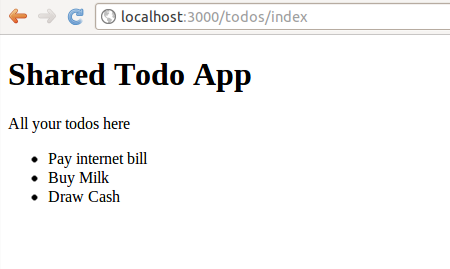


Figure: Todo app without bootstrap.css

1. We are going to use the Bootstrap gem by Sehjunak on Github -

<https://github.com/seyhunak/twitter-bootstrap-rails/>.

Install bootstrap gem by adding the following lines to Gemfile and then doing bundle install

Gemfile

gem 'less-rails'

gem 'twitter-bootstrap-rails'

You can add the line anywhere in the Gemfile

Console

user\_1@codelearn:~/shared\_todo\_app$bundle install

Now the Bootstrap gem is installed and ready to use.

1. As per instructions at [https://github.com/seyhunak/twitter-bootstrap-](https://github.com/seyhunak/twitter-bootstrap-rails/) [rails/](https://github.com/seyhunak/twitter-bootstrap-rails/), execute the below command in your Rails app :-

Console

user\_1@codelearn:~/shared\_todo\_app$ rails g bootstrap:install insert app/assets/javascripts/application.js

create app/assets/javascripts/bootstrap.js.coffee

create app/assets/stylesheets/bootstrap\_and\_overrides.css.less gsub app/assets/stylesheets/application.css

gsub app/assets/stylesheets/application.css

The app should be back to its old self now.

Warning! You may have to restart the rails server for everything to work properly.

### What is Ruby Gem

From the above exercise, it is clear that what we could do manually following many steps, Ruby gems lets us do it all in very few steps inside our Ruby/Rails app.

Quite a few awesome Rails developers are doing awesome job world over. If their work is independent, modular & it has a usecase for others (like automating addition of Bootstrap files in the project); they create a Ruby gem out of it.

While Bootstrap gem is relatively simpler gem that we saw here, some gems like [devise](https://github.com/plataformatec/devise) for authentication have been amazingly useful to Rails developers.

Teaser In our Rails advanced course slated to be released shortly, we talk about how to share your to-do list app with friends. Using devise you can very quickly incorporate invitation as well as authentication part without needing to write a single line of code.

### Making index page, the home page.

In your current app, '/' or the root URL still shows public/index.html. Since the main page of our app is the index page, lets change the routes to show it on the root URL.

1. Remove the file public/index.html

Console

rm public/index.html

1. Add a line to routes.rb (anywhere)

config/routes.rb

root :to => 'todos#index'

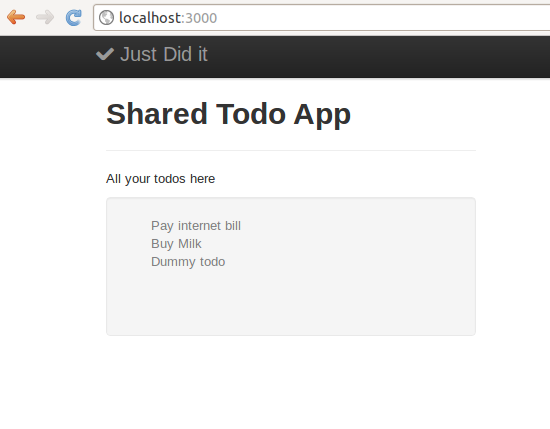
Retaining the line get "todos/index" in the file, will allow you to access the app using both the URLs . We will use both interchangeably in the lessons.

1. localhost:3000/todos/index
2. localhost:3000

The above 2 changes will make the todos app your home page, as shown in the below screenshot (notice the URL).

Figure: Todo app as the home page, as against the default rails app

[← Previous](#_bookmark21) [Next →](#_bookmark43)

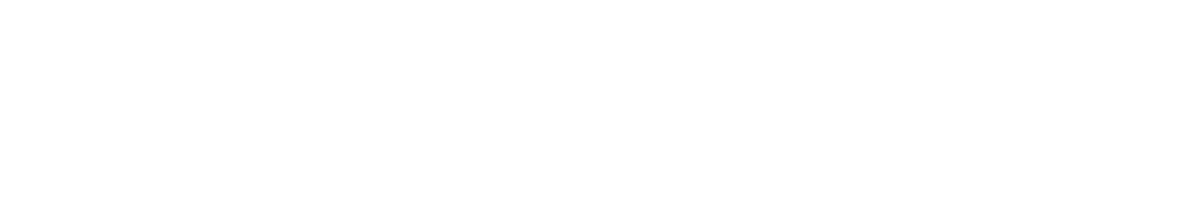


© Codelearn 2012

[**Follow**](#_bookmark43) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark43) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Remove bootstrap.css, incorporate Gem, create fixed layout

[Module 1. Getting started with Rails](#_bookmark43)



[← Previous](#_bookmark41) [Next →](#_bookmark49)

[Module 2. Starting with to-do list app](#_bookmark44)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

|  |
| --- |
| [1. Designing to-do list](#_bookmark18) [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30) [todos](#_bookmark30) |
| [Todos from](#_bookmark32) [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41) [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43) [fixed layout](#_bookmark43) |

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

Implement the steps in the parent lesson '[Bootstrap CSS via Ruby Gem](#_bookmark41)'. Remove bootstrap.css & bootstrap.js from shared\_todo\_app. Add the Bootstrap gem. Do bundle install followed by rails g bootstrap:install .

This task will take you deeper into the Bootstrap gem that we implemented. We will also see that it is substantially less work doing something the gem way.

1. Create a new layout bootstrap as per [https://github.com/seyhunak/twitter-](https://github.com/seyhunak/twitter-bootstrap-rails/) [bootstrap-rails/](https://github.com/seyhunak/twitter-bootstrap-rails/). Follow the command below.

Console

user\_1@codelearn.org~/shared\_todo\_app$ rails g bootstrap:layout bootstrap fluid create app/views/layouts/bootstrap.html.erb

Note The layout name is 'bootstrap'. If you replace it with 'application', it will end up overriding the existing app/views/layouts/application.html.erb

1. Create a new view file app/views/todos/index-bootstrap.html.erb.

Console

user\_1@codelearn.org~/shared\_todo\_app$ touch app/views/todos/index-bootstrap.html.e rb

1. Change index action in todos controller to use the new bootstrap layout & render the newly created index-bootstrap.html.erb.

[Module 3. Add, delete, mark &](#_bookmark43) [unmark todos](#_bookmark43)

Hint You would need to use render method. Refer '[Add Twitter Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)' to read more about how to use 'render'.

1. Edit bootstrap.html.erb & index-bootstrap.html.erb to get exactly similar look as with application.html.erb & index.html.erb.

You would notice that most of the hand created stuff for application.html.erb like navbar is already present in bootstrap.html.erb. You would need to remove navbar links & the sidebar. It would require you some level of HTML skills to do so.

1. Once you are done with above tasks, hit 'Run tests' below. Once the tests pass, overwrite application.html.erb & index.html.erb with bootstrap.html.erb

& index-bootstrap.html.erb respectively.

Console

mv app/views/layouts/bootstrap.html.erb app/views/layouts/application.html.erb mv app/views/todos/index-bootstrap.html.erb app/views/todos/index.html.erb

Note Execute above commands only when all the tests are passing. If you are unable to complete this task, move on to the subsequent lesson with the old application.html.erb & index.html.erb

Test Output

Click 'Run test' button above to verify if your task solution is right.

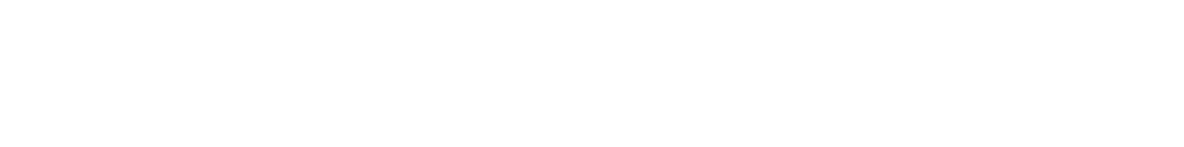
[← Previous](#_bookmark41) [Next →](#_bookmark49)

© Codelearn 2012

[**Follow**](#_bookmark46) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark46) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Toggle todos status

[Module 1. Getting started with Rails](#_bookmark46)



[← Previous](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete)



[Next →](#_bookmark46)

[Module 2. Starting with to-do list app](#_bookmark46)

[Module 3. Add, delete, mark &](#_bookmark47) [unmark todos](#_bookmark47)

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In this task, you will enhance the 'mark todo complete' feature to include toggling the todo status. Todos that are not completed, if marked & submitted, will get completed. Todos that are completed will go to incomplete state.

1. Start with the app built at the end of the previous task '[Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete)'.
2. Change the 'Complete Todos' button name to 'Toggle Todos'.
3. Edit 'complete' action in Todos controller. Check if t.completed == true then
   1. pdate\_attribute(:completed, false) else t.update\_attribute(:completed, true) .

Test Output

|  |
| --- |
| [1. Add todo from the](#_bookmark49) [view](#_bookmark49) |
| [Form to add todo](#_bookmark52) |
| [form\_for instead of](#_bookmark55) [form\_tag](#_bookmark55) |
| [2. Mark todo complete](#_bookmark58) |
| [Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) |
| [Toggle todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/2/Toggle-todos-status) |
| [3. Edit & delete todos](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/3/Coming-soon) |
| [4. Ajaxifying operations](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/4/Coming-soon) |

Click 'Run test' button above to verify if your task solution is right.

[← Previous](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete)

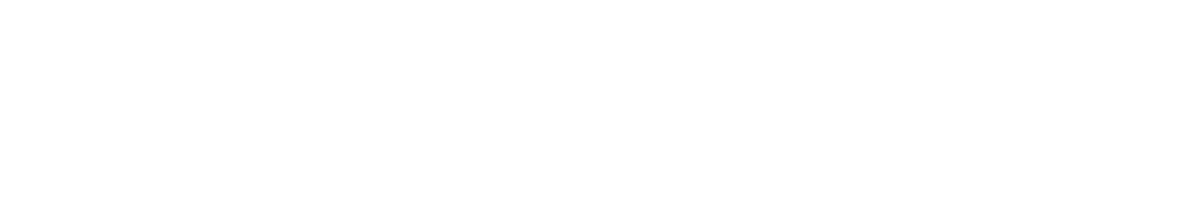
[Next →](#_bookmark46)

© Codelearn 2012

[**Follow**](#_bookmark49) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark49) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Rails forms, params, attr\_accessible, validation

[Module 1. Getting started with Rails](#_bookmark49)



[← Previous](#_bookmark43) [Next →](#_bookmark52)

[Module 2. Starting with to-do list app](#_bookmark49)

[Module 3. Add, delete, mark &](#_bookmark50) [unmark todos](#_bookmark50)



|  |
| --- |
| [1. Add todo from the](#_bookmark49)  [view](#_bookmark49) |
| [Form to add todo](#_bookmark52) |
| [form\_for instead of](#_bookmark55) [form\_tag](#_bookmark55) |
| [2. Mark todo complete](#_bookmark58) |
| [Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) |
| [Toggle todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/2/Toggle-todos-status) |
| [3. Edit & delete todos](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/3/Coming-soon) |
| [4. Ajaxifying operations](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/4/Coming-soon) |

In the previous module, you created a to-do list app which listed all the todos that you added from the Rails console to the Todo model. You also added a link to delete the last todo. You also added Twitter Bootstrap CSS for the app to look 'nice'.

That is a good start but the app is not usable yet. You are able to add the todos from the backend (Rails console). It is time to add, delete, mark & unmark the todos from the view (browser).



Teaser In our paid Rails advanced course, you will be able to 'share' the to-do list app with your friends. You can invite set of friends who can login, log to- dos & mark/unmark them.



[Module3](#_bookmark51) [Lesson 1 Lesson 2 Lesson 3](#_bookmark49)

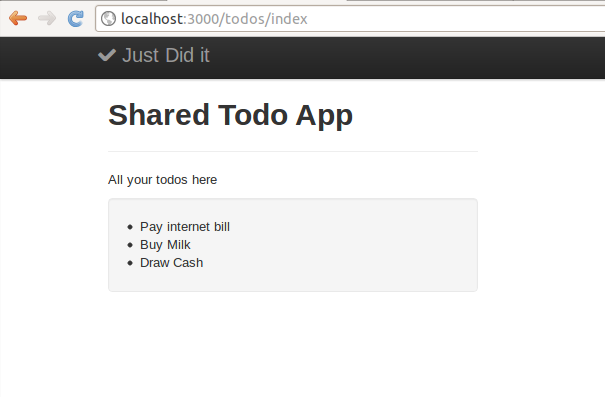


Figure : How the app look so far (completion of previous Module).

This lesson teaches you how to add a todo from the view using forms in Rails. It also covers form & model validation.

### Planning the 'add todo from the view' feature

Before we step out to implement a new feature, it is always good to have a plan. As in the lesson '[Designing to-do list app](#_bookmark18)', where we planned the basic app structure, a similar plan for each feature is a good thing.

Teaser In the Rails advanced course, we will see how to convert these feature plans below to rspec tests. The approach is called TDD.

So the plan is as follows

* + 1. A form with a text box to enter todo\_item name & a submit button to send data to the url /todos/add.
    2. A route entry to match the /todos/add to a new action 'add' inside Todos controller.
    3. The action 'add' which creates a new todo in the database using Todo model & then redirects the control to index action.

### Implementing the planned feature

##### The form in the view

Add the below code below the line 'All your todos here'

app/views/todos/index.html.erb

* 1. <p>All your todos here</p>
  2. <%= form\_tag("/todos/add", :method=>"post") do %>
  3. <%= text\_field\_tag(:todo\_text) %>

1. <%= submit\_tag("Add todo", :class=>"btn") %>
2. <% end %>

Those who are familiar with how 'form' looks in HTML would be able to understand the above code better. The code when processed by Rails churns out a nice HTML form as below.

<form accept-charset="UTF-8" action="/todos/add" method="post">

<input id="todo\_text" name="todo\_text" type="text">

<input name="commit" type="submit" value="Add todo">

</form>

##### Adding route

The URL /todos/add need to be mapped to the add action in Todos controller.

config/routes.rb

get "todos/index"

match "todos/add" => "todos#add", :via => :post

##### Adding 'add' action in Todos controller

app/controllers/todos\_controllers.rb

class TodosController < ApplicationController

def index

.

.

.

end

def delete

.

.

end

def add

Todo.create(:todo\_item => params[:todo\_text])

redirect\_to :index

end end

##### Making sure Todo.create works

The changes done till now will break when the add action will try executing 'Todo.create' (reason explained later). You need to add following lines in the model file.

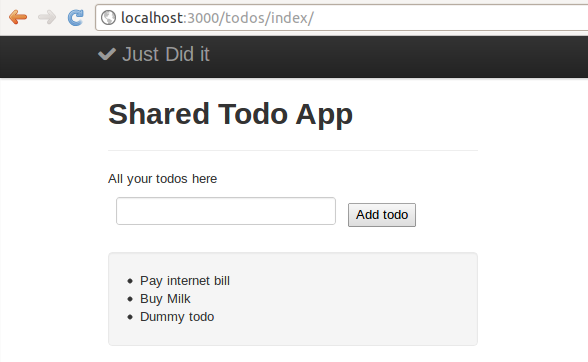
app/models/todo.rb

* 1. class Todo > ActiveRecord::Base
  2. attr\_accessible :todo\_item
  3. end

The steps above should get you the form to add todos working. The details of each steps are discussed in the subsequent section.

Figure: Todo list app with the form at the end of above steps.

### The finer details



##### Dissecting form\_tag helper function

Helper functions are introduced in lesson '[Add Twitter Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)' where we used link\_to. form\_tag as the name suggest, lets the user create a form in the view. You may read more about HTML forms at <http://www.w3schools.com/html/html_forms.asp>

Note form\_for is the more refined version & widely used as against form\_tag. You will learn to use form\_for in the subsequent task.

The first argument to form\_tag is the url to which data is sent - /todos/add . It takes an argument :method with value "post".

<%= form\_tag("/todos/add", :method=>"post") do %>

Similarly there is text\_field\_tag & submit\_tag helper functions which creates text input & submit button respectively.

As mentioned in lesson '[Add Twitter Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)', the brackets '()' are optional in Rails functions. So

form\_tag "/todos/add", :method => "post"

is completely acceptable. You may read more about form helper functions [here](http://guides.rubyonrails.org/form_helpers.html)

The HTTP method that is get/post/put need to be defined for the routes & hence you see

:via => "post"

in the route entry.

##### How params[:todo\_text] got the data ?

Note the line

Todo.create(:todo\_item => params[:todo\_text])

in 'add' action that has been added.

params is a reserved variable that is available inside controller actions. It holds the data that is passed over from the url as key value pair.

Since the text field for todo\_item has the name 'todo\_text'

<%= text\_field\_tag(:todo\_text) %>

params[:todo\_text] has the value of the input field.

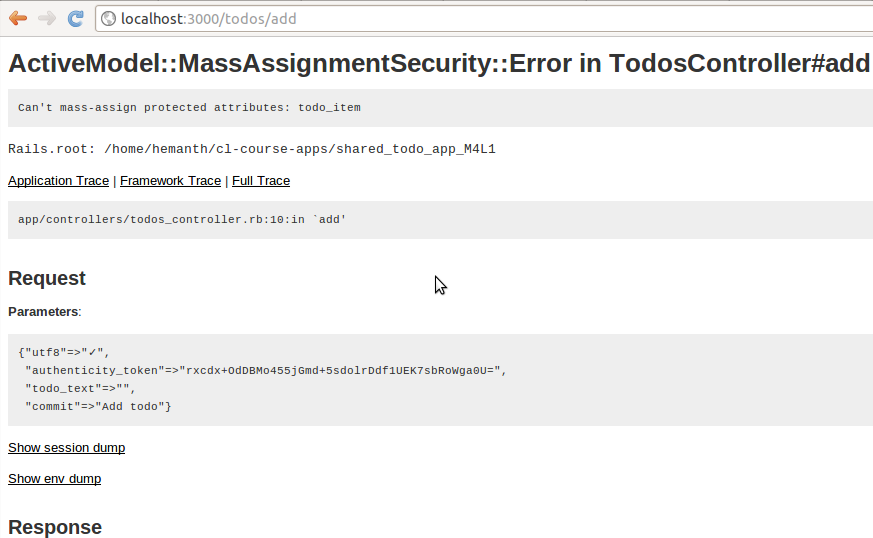
##### Why attr\_accessible ?

Another new thing here is using :attr\_accessible :todo\_item inside app/models/todo.rb. The line lets controller function mass assign value to the todo\_item while creating a new entry in the database using Todo.create.

Without the line, you will see the error as shown below.

Figure: Screenshot when attempting to add empty todo.

### Model validation & flash variable



The app accepts empty todos. Try hitting on 'Add Todo' button with text box empty.

Lets remove this bug by adding model validation

##### Adding model validation

app/models/todo.rb

* 1. class Todo > ActiveRecord::Base
  2. attr\_accessible :todo\_item
  3. validates :todo\_item, presence: true

4. end

validates is a Rails model validation helper. You can read more about it [here](http://guides.rubyonrails.org/active_record_validations_callbacks.html#validation-helpers). You can test the above changes inside Rails console as below.

Rails console

1.9.3p194 :002 > Todo.create(:todo\_item => "") (0.1ms) begin transaction

(0.1ms) rollback transaction

=> #

Observe "rollback transaction above.

1.9.3p194 :003 > t = Todo.new(todo\_item: "")

=> #

1.9.3p194 :004 > t.save (0.1ms) begin transaction

(0.1ms) rollback transaction

=> false

1.9.3p194 :005 > t.valid?

=> false

'valid?' is a method/function used by Rails to check if an object is valid. More about the method [here](http://guides.rubyonrails.org/active_record_validations_callbacks.html#validation-helpers)

##### Catching error in the controller

app/controllers/todos\_controller.rb

* 1. def add
  2. todo = Todo.create(:todo\_item => params[:todo\_text])
  3. if !todo.valid?

1. flash[:error] = todo.errors.full\_messages.join("<br>").html\_safe
2. end
3. redirect\_to :index
4. end

The line

if !todo.valid?

is an if statement with a condition. todo.valid? returns true if todo object is valid. Putting a '!' before it inverts the logic. Hence control comes inside the if statement if the todo is invalid.

flash stores the value that is available in the next HTTP request unlike instance variable @todo\_item. The redirect at the of the add action makes the browser send a new request to /todos/index . The index action & consequently the index.html.erb has access flash variable value that got stored by the add action. More about flash [here](http://guides.rubyonrails.org/action_controller_overview.html#the-flash).

todo.errors[] is an array of errors. 'full\_messages', 'join' & 'html\_safe' are methods which you do not have to worry about now.

##### Showing error on the view.

app/views/todos/index.html.erb

* 1. <p>All your todos here</p>
  2. <%= form\_tag("/todos/add", :method => "post") do %>
  3. <%= text\_field\_tag(:todo\_text) %>

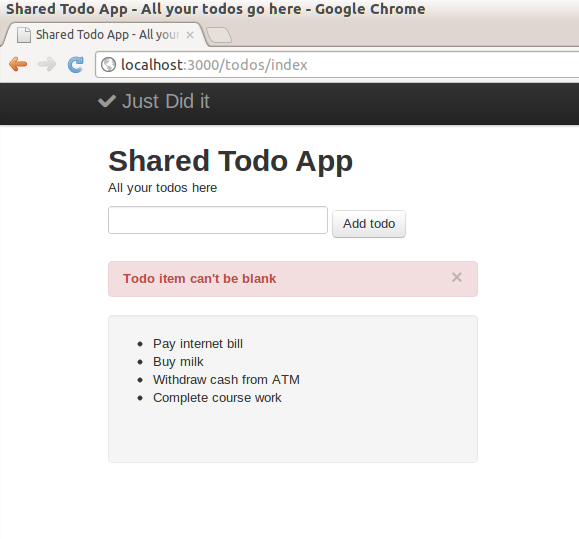
1. <%= submit\_tag("Add todo", :class => "btn ") %>
2. <% end %>
3. <% if !flash[:error].nil? %>
4. <div class="alert alert-error">
5. <button type="button" class="close" data-dismiss="alert">×</button>
6. <strong><%= flash[:error] %></strong>
7. </div>
8. <% end %>

Like 'valid?', nil? is a Rails method to check if an object is nil (undefined). flash[:error] contains the error value (if any) set by add action in the previous HTTP request (/todos/add).

<div class="alert alert-error"> ... </div> is a way to show alert box in Twitter Bootstrap CSS. The class 'alert-error' gives the alert box the red color. More about Twitter Bootstrap alert box [here](http://twitter.github.com/bootstrap/components.html#alerts)

Figure: Screenshot when attempting to add empty todo.

[← Previous](#_bookmark43) [Next →](#_bookmark52)

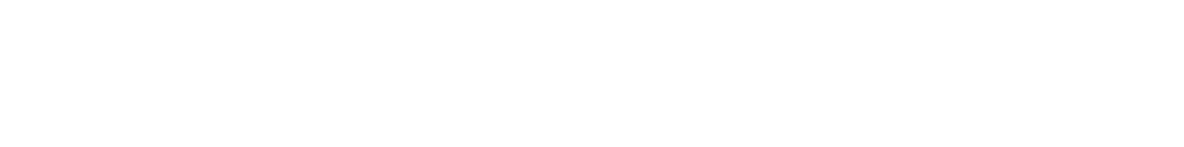


© Codelearn 2012

[**Follow**](#_bookmark52) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark52) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - Implement adding todo

[Module 1. Getting started with Rails](#_bookmark52)



[← Previous](#_bookmark49) [Next →](#_bookmark55)



[Module 2. Starting with to-do list app](#_bookmark52)

[Module 3. Add, delete, mark &](#_bookmark53) [unmark todos](#_bookmark53)

|  |
| --- |
| [1. Add todo from the](#_bookmark49) [view](#_bookmark49) |
| [Form to add todo](#_bookmark52) |
| [form\_for instead of](#_bookmark55) [form\_tag](#_bookmark55) |
| [2. Mark todo complete](#_bookmark58) |
| [Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) |
| [Toggle todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/2/Toggle-todos-status) |
| [3. Edit & delete todos](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/3/Coming-soon) |
| [4. Ajaxifying operations](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/4/Coming-soon) |

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

In this task, you will need to implement what is the taught in the parent lesson

'[Add todo from the view](#_bookmark49)'.

1. Start from the app that built at the end of the last Module.
2. Follow the parent lesson. Add a form with an input field to accept todos. Add 'add' action in Todos controller. Create routes entry for 'add' action. Also implement validation as mentioned in the later part of the lesson. Below is the screenshot of the expected app at the end of the task.

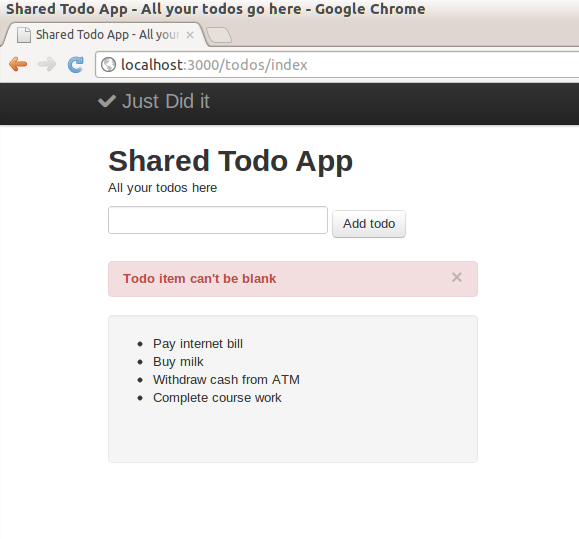


Figure: Screenshot of todo app showing an error for empty todo submission.

Test Output

Click 'Run test' button above to verify if your task solution is right.

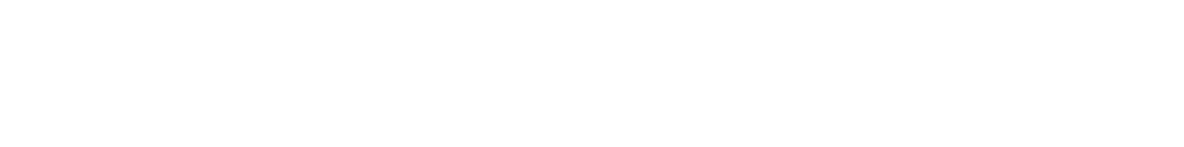
[← Previous](#_bookmark49) [Next →](#_bookmark55)

© Codelearn 2012

[**Follow**](#_bookmark55) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark55) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Task - How to use form\_for

[Module 1. Getting started with Rails](#_bookmark55)



[← Previous](#_bookmark52) [Next →](#_bookmark58)



[Module 2. Starting with to-do list app](#_bookmark55)

[Module 3. Add, delete, mark &](#_bookmark56) [unmark todos](#_bookmark56)

|  |
| --- |
| [1. Add todo from the](#_bookmark49) [view](#_bookmark49) |
| [Form to add todo](#_bookmark52) |
| [form\_for instead of](#_bookmark55) [form\_tag](#_bookmark55) |
| [2. Mark todo complete](#_bookmark58) |
| [Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) |
| [Toggle todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/2/Toggle-todos-status) |
| [3. Edit & delete todos](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/3/Coming-soon) |
| [4. Ajaxifying operations](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/4/Coming-soon) |

Note Use [CodeLearn Playground](http://www.codelearn.org/apps/code_play) to complete the task below.

If you have problems completing the task or you need assistance, reach us at [hemanth@codelearn.org](mailto:hemanth@codelearn.org)

This task will make you hands on with form\_for - a more commonly & widely used Rails helper function for showing forms on the view.

1. Start from the app you built in the previous task '[Form to add todo](#_bookmark52)'. It should have a form implemented using form\_tag, an add action in Todos controller & validation to stop empty todo submission plus showing it on the browser through Twitter Bootstrap alert box.
2. Replace the form\_tag code with form\_for code in

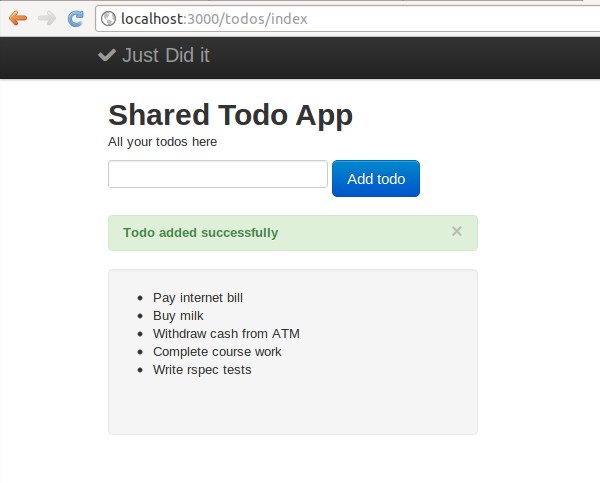
app/views/todos/index.html.erb. Check for form\_for explaination at the end.

1. Add a small feature. When you add a todo, a display message showing successfull submission should be displayed like the screenshot below.

Hint Look at the code on how flash[:error] was set. Set flash[:success] for the case of success & show the value on the view.

Figure: Screenshot of todo app, with "Todo added successfully message"

##### Explaining form\_for helper function



Like form\_tag, form\_for serves the same purpose of showing a form on the view. The major difference is - form\_for works with a model object.

For eg. if we create a model object @new\_todo as @new\_todo = Todo.new , we can now use form\_for @new\_todo, :url => {:action => "add" } do |f| in the view. Note that we used form\_tag "/todos/add", :method => :post earlier.

While form\_tag needed us to mention the url & the method, form\_for requires to mention the model object (@new\_todo) & the url.

Note form\_for decides the method based on the model object. If @new\_todo is empty (created by Todo.new), the method is POST. If @new\_todo contained a Todo @new\_todo = Todo.find(1) , the method is set as PUT.

form\_for encapsulates all its child element. text\_field\_tag(:text\_item) becomes

f.text\_field(:text\_item) . Note how 'f' has appeared (highlighted below).

<%= form\_for @new\_todo, :url => {:action => "add" } do |f| %>

# put the child elements - text field & the submit button

<% end %>

The implication is, inside the action add which receives the form data post submission, the data becomes available as params[*name of the model of which*

*@new\_todo is object*][*name of the element in the form*] . In our case, it is

params[:todo][:text\_item] .

Test Output

Click 'Run test' button above to verify if your task solution is right.

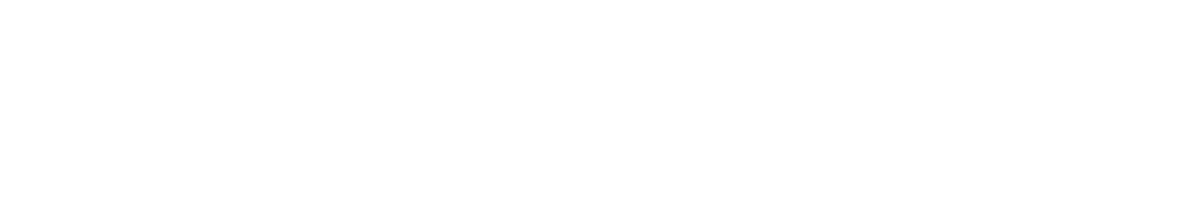
[← Previous](#_bookmark52) [Next →](#_bookmark58)

© Codelearn 2012

[**Follow**](#_bookmark58) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark58) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

# Rails tutorial - Lesson - Rails migration, form\_for & checkboxes

[Module 1. Getting started with Rails](#_bookmark58)



[← Previous](#_bookmark55) [Next →](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete)

[Module 2. Starting with to-do list app](#_bookmark58)

[Module 3. Add, delete, mark &](#_bookmark59) [unmark todos](#_bookmark59)



|  |
| --- |
| [1. Add todo from the](#_bookmark49)  [view](#_bookmark49) |
| [Form to add todo](#_bookmark52) |
| [form\_for instead of](#_bookmark55) [form\_tag](#_bookmark55) |
| [2. Mark todo complete](#_bookmark58) |
| [Marking todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) [complete](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete) |
| [Toggle todos](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/2/Toggle-todos-status) |
| [3. Edit & delete todos](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/3/Coming-soon) |
| [4. Ajaxifying operations](http://www.codelearn.org/ruby-on-rails-tutorial/3/Introducing-forms%2C-migrations-%26-ajax-in-Rails/4/Coming-soon) |

In the previous lesson '[Add todo from the view](#_bookmark49)', you learnt about how to add todos from the browser using forms. So you have an app as shown in the tab 'Your app at the start' by now.

This chapter talks about implementing 'mark todo complete' feature. You will learn to use checkboxes, its handling in Rails and how to add database fields using Rails migration.



[Your app at the start](#_bookmark60) [Your app at the end](#_bookmark58)

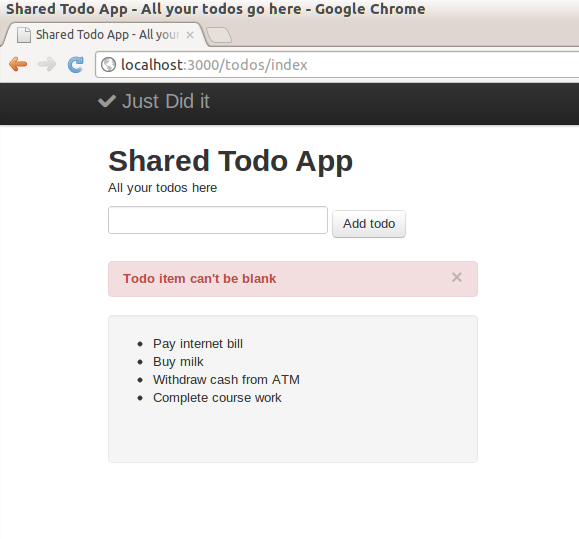


Figure : Screenshot of the app at start of lesson2.

### Planning 'mark todo complete' feature

As usual, we start with a plan as to how the feature will be implemented.

* 1. Lets show checkboxes next to each todo item. Checking a few & then hitting on a button saying 'Complete Todos' should be good. Checkboxes will let us mark multiple todos complete at the same time.
  2. On hitting 'Complete Todos', data is sent to /todos/complete/ via POST (refer [Designing to-do list app](#_bookmark18)). We need complete action inside Todos controller(app/controllers/todos\_controllers.rb) to be able to handle the data, update it in the database using Todo model.
  3. The new complete action needs to update the status of each todo item from pending to completed. But wait, we do not have the status field for the Todo model yet. Good time to introduce it through migration.
  4. Getting back to the complete action, as discussed in Module 2 - Lesson 2, it should redirect to index action to show all the todos. But wait, there are todos with different status now. In the view, we need to show the complete todos as striked out.

Now things look better. Lets start implementing.

### Implementing the feature.

##### Form with checkboxes & 'Complete todos' button

Remove the code striked out & add the code in the view file.

app/views/todos/index.html.erb

<p>All your todos here</p>

.

# Form to add a todo here followed by flash[:error] & flash[:success] code

.

<div class="well">

~~<ul>~~

~~<% @todo\_items.each do |t| %>~~

~~<li><%= t.name %></li>~~

~~<% end %>~~

~~</ul>~~

<%= form\_tag("/todos/complete/", :method => "post") do %>

<ul style="list-style-type:none;">

<% @todo\_items.each do |t| %>

<li> <%= check\_box\_tag "todos\_checkbox[]",t.id %> <%= t.todo\_item %> </li>

<%end%>

</ul>

<%= submit\_tag("Complete Todos", :class=>"btn btn-success") %>

<%end %>

This should show a todos with checkboxes on the browser.

##### Handling form data inside 'complete' action

Mapping the url to the action. Add the below line anywhere in config/routes.rb

file.

config/routes.rb

match 'todos/complete' => 'todos#complete', :via => :post

Add the complete action in the Todos controller.

app/controllers/todos\_controllers.rb

class TodosController < ApplicationController

def index

.

.

.

end

def delete

.

.

end

def add

.

.

end

def complete

params[:todos\_checkbox].each do |check|

todo\_id = check

t = Todo.find\_by\_id(todo\_id)

#code to update the status here

end

redirect\_to "/todos/index"

end

end

This is an incomplete function. We do not have the status field in the Todo model yet. We will come to the function after step 3.

##### Introducing status field in Todo model through migration

Console

user\_1@codelearn:~/shared\_todo\_app$ rails generate migration AddCompletedToTodo invoke active\_record

create db/migrate/20120805093803\_add\_completed\_to\_todo.rb

Open the migration file & add the highlighted code.

Note The actual name of the file will be different for you. Pick the file name from the output of the migration commmand above.

db/migrate/20120805093803\_add\_completed\_to\_todo.rb

class AddCompletedToTodo < ActiveRecord::Migration def change

add\_column :todos, :completed, :boolean end

end

Execute the command below to update the database schema to add the

:completed field.

Console

user\_1@codelearn:~/shared\_todo\_app$ rake db:migrate

== AddCompletedToTodo: migrating =============================================

-- add\_column(:todos, :completed, :boolean)

-> 0.0011s

== AddCompletedToTodo: migrated (0.0012s) ====================================

We need to add 'attr\_accessible' for the field to be able to update it from the controller.

app/models/todo.rb

class Todo > ActiveRecord:Base attr\_accessible :todo\_item, :completed validates :todo\_item, presence: true

end

##### Updating status in 'complete' action, showing status in 'index' view.

app/controllers/todos\_controller.rb

class TodosController < ApplicationController

def index

.

.

.

end

def delete

.

.

end

def add

.

.

end

def complete

params[:todos\_checkbox].each do |check|

todo\_id = check

t = Todo.find\_by\_id(todo\_id)

t.update\_attribute(:completed, true)

end

redirect\_to "/todos/index"

end

app/views/todos/index.html.erb

<%= form\_tag("/todos/complete/", :method => "post") do %>

<ul style="list-style-type:none;">

<% @todo\_items.each do |t| %>

~~<li> <%= check\_box\_tag "todos\_checkbox[]",t.id %> <%= t.todo\_item %> </li>~~

<% if t.completed == true %>

<li style="color:grey;"> <%= check\_box\_tag "todos\_checkbox[]",t.id %> <stri ke><%= t.todo\_item %></strike> </li>

<% else %>

<li> <%= check\_box\_tag "todos\_checkbox[]",t.id %> <%= t.todo\_item %> </li>

<% end %>

<%end%>

</ul>

<%= submit\_tag("Complete Todos", :class=>"btn btn-success") %>

<%end %>

The app should look like below if you have got all the steps right.

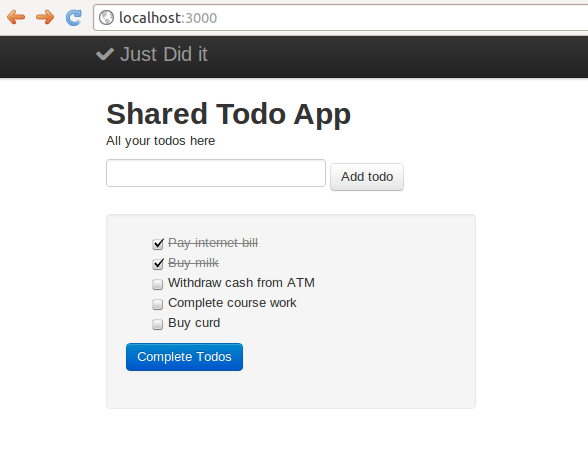


Figure: Todo app with checkboxes to mark todos complete. Completed todos are striked out.

### The finer details

##### Rails, form\_tag & check\_box\_tag.

<%= form\_tag("/todos/complete/", :method => "post") do %>

form\_tag is already introduced in Lesson 1 '[Add todo from the view](#_bookmark49)'. The behaviour is the same as showed in Lesson1.

/todos/complete/ denotes the URL to which data is sent to.

:method => :post means you are sending data. get means you are getting data from the application, in very simplistic terms.

In HTML, the checkboxes have same 'name' attribute. <%= check\_box\_tag

"todos\_checkbox[]",t.id %> says that every checkbox gets the same name 'todos\_checkbox[]'. The value each checkbox holds is t.id which is the id of the respective todo.

In controller, Rails combine the values of the selected checkboxes in an array. For eg. - if two checkboxes with values 1 & 2 are checked respectively, params[:todos\_checkbox] will look like [1, 2] .

##### Rails migration

rails generate migration AddCompletedToTodo

The name 'AddCompletedToTodo' is used by Rails to name the file. The name of the class inside the file also has the same name. Apart from it, the name does not have much significance.

The change function inside the migration file gets executed during rake

db:migrate

add\_column :todos, :completed, :boolean

The above line instructs Rails to add a column 'completed' to table 'todos'. The column is type boolean. Similar to 'add\_column', there are other migration methods like 'remove\_column', 'change\_column' etc. You may read more about migration methods at <http://guides.rubyonrails.org/migrations.html>

Note 'rake db:migrate' only add a migration once. It uses the timestamp in the migration file to determine which are the newly created migrations. For eg. 20120805093803 is the timestamp in the migration file created in this example.

Tip If you created a migration file, did 'rake db:migrate' & realized that you need to change something in the migration file; doing a 'rake db:migrate' again will not help. You would need to do rake db:rollback , change content of the migration file & do 'rake db:migrate' again.

If you can afford to loose the existing data in the database, you can drop the

database & reapply all migrations from scratch by following commands.

rake db:drop rake db:migrate

##### update\_attribute

t.update\_attribute(:completed, true)

The above line is added in the complete action. As the name suggest, the method available to the model object updates the field value 'completed' to true. To see the function in action, try the commands below.

Console

rails c

Rails Console

t = Todo.first

Todo Load (0.2ms) SELECT "todos".\* FROM "todos" LIMIT 1

=> #<Todo id: 1, todo\_item: "Pay internet bill", created\_at: "2012-08-23 11:24:00"

, updated\_at: "2012-08-23 11:24:00", completed: nil>

* + 1. p194 :002 > t.update\_attribute(:completed, true) (0.1ms) begin transaction

(0.7ms) UPDATE "todos" SET "completed" = 't', "updated\_at" = '2012-09-02 17:28: 58.381107' WHERE "todos"."id" = 1

(100.5ms) commit transaction

=> true

[← Previous](#_bookmark55) [Next →](http://www.codelearn.org/ruby-on-rails-tutorial/tasks/3/2/Rails-migration%2C-form_for-%26-checkboxes/1/Marking-todos-complete)

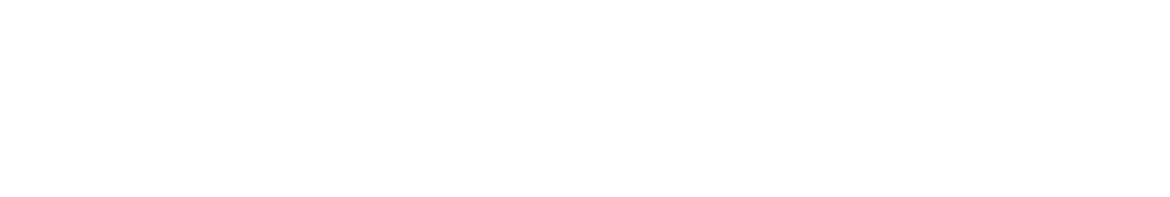
© Codelearn 2012

[**Follow**](#_bookmark61) [**@codelearn**](https://twitter.com/intent/follow?original_referer&amp;region=follow_link&amp;screen_name=codelearn&amp;source=followbutton&amp;variant=2.0) [Like](#_bookmark61) [Home](http://www.codelearn.org/launch) [Philosophy](http://www.codelearn.org/ruby-on-rails-tutorial/Codelearn-philosophy) [Codelearn Playground](http://www.codelearn.org/apps/code_play) [Blog](http://www.codelearn.org/blog)

Rails tutorial - Lesson - Creating routes, controllers & views

| ruby tutorial

[Module 1. Getting started with Rails](#_bookmark61)



[← Previous](#_bookmark18) [Next →](#_bookmark24)

[Module 2. Starting with to-do list app](#_bookmark62)

|  |
| --- |
| [1. Designing to-do list](#_bookmark18)  [app](#_bookmark18) |
| [2. Todos from controller](#_bookmark61) |
| [Display one todo](#_bookmark24) |
| [Display multiple](#_bookmark27) [todos](#_bookmark27) |
| [3. Model to store/fetch](#_bookmark30)  [todos](#_bookmark30) |
| [Todos from](#_bookmark32)  [database](#_bookmark32) |
| [Delete last todo](#_bookmark35) |
| [4. Add Twitter](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline)  [Bootstrap CSS](http://www.codelearn.org/ruby-on-rails-tutorial/2/Building-to-do-list-web-app/4/Views%2C-layout%2C-helper-methods-%26-assets-pipeline) |
| [Bootstrap CSS with](#_bookmark38) [navbar](#_bookmark38) |
| [Use modal & helper](#_bookmark21) [functions](#_bookmark21) |
| [5. Bootstrap CSS via](#_bookmark41)  [Ruby Gem](#_bookmark41) |
| [Bootstrap Gem,](#_bookmark43)  [fixed layout](#_bookmark43) |

[Module 3. Add, delete, mark &](#_bookmark61) [unmark todos](#_bookmark61)

In the last lesson - [Designing to-do list app](#_bookmark18), you designed the to-do list app which we are building as part of this tutorial.

In this lesson, you will learn how to use Rails controller by learning to add static to-dos.

#### Default rails app - shared\_todo\_app

As per the design in the last lesson - [Designing to-do list app](#_bookmark18), the URL

/todos/index has to return a page listing all the todos.

As the first step, let's start with a new rails app - shared\_todo\_app

Console

user\_1@codelearn:~$ rails new shared\_todo\_app user\_1@codelearn:~$ cd shared\_todo\_app

#Edit Gemfile to uncomment rubyracer gem user\_1@codelearn:~/shared\_todo\_app$ bundle install user\_1@codelearn:~/shared\_todo\_app$ rails server

Note The detailed steps are mentioned at [Zero to app in 5 steps](#_bookmark0). The default rails app should show up inside App Output as below

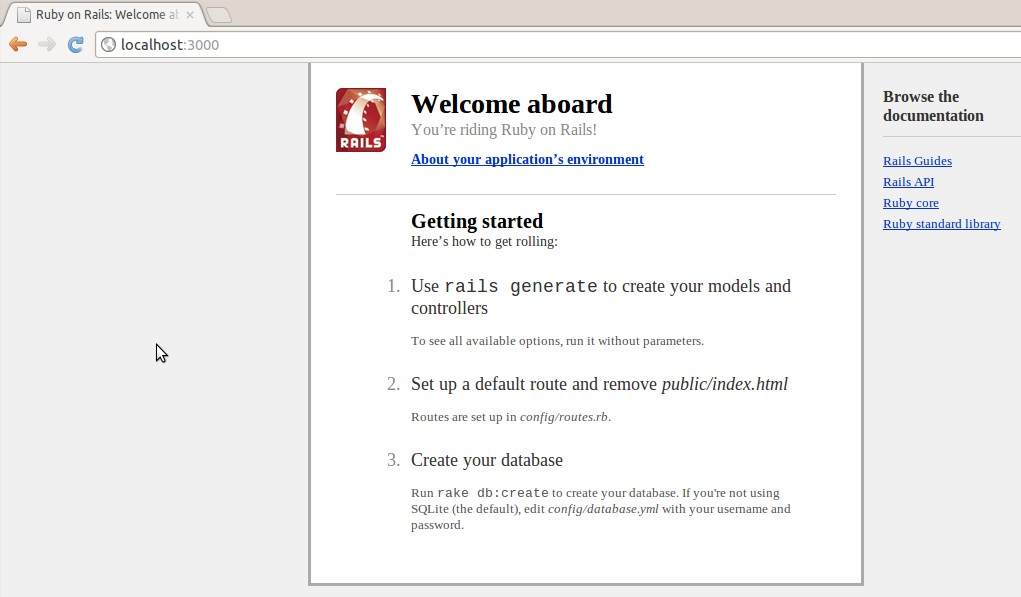


Figure: Default rails app

#### Generating Todos controller

In last lesson - [Designing to-do list app](#_bookmark18), you saw that to handle the URL

/todos/index, you need an index action in the Todos. The index action displays all to-dos.

Note Controller is rails way of handling server side functionality for a URL. The following command generates a Todos controller with an index action.

Console

user\_1@codelearn:~/shared\_todo\_app$ rails generate controller Todos index

Command Output

create app/controllers/todos\_controller.rb route get "todos/index"

invoke erb

create app/views/todos

create app/views/todos/index.html.erb invoke test\_unit

create test/functional/todos\_controller\_test.rb invoke helper

create app/helpers/todos\_helper.rb invoke test\_unit

create test/unit/helpers/todos\_helper\_test.rb invoke assets

invoke coffee

create app/assets/javascripts/todos.js.coffee invoke scss

create app/assets/stylesheets/todos.css.scss

Controller name is always plural as per convention. Todos instead of Todo, Tweets instead of Tweet, Blogs instead of Blog.

The rails generate controller command created a lot of files and directories. We are interested in two of them for now :-

app/controllers/todos\_controller.rb app/views/todos/index.html.erb

Inside Rails app, the controller file is placed inside app/controllers directory. In Rails every file has its place and it is uniform across all rails apps.

app/controllers/todos\_controller.rb

* + - 1. class TodosController < ApplicationController
      2. def index

3.

4. end

5. end

The TodosController is a class. There is an empty method/function index inside the class. The function is also called index action in the Todos controller.

This action serves the url /todos/index when accessed. Even though the index action is an empty method, it fetches the file

/apps/views/todos/index.html.erb & displays it on the browser.

Note The file app/views/todos/index.html.erb is referred to as view file in Rails. This is part of the Views in Rails MVC architecture.

#### Touching views

Run the rails server to see the output of the app.

Console

user\_1@codelearn:~/shared\_todo\_app$ rails server

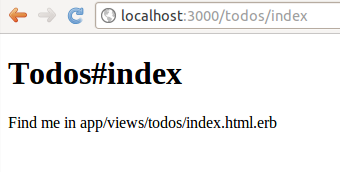
The default html page that gets rendered for the URL /todos/index is

app/views/todos/index.html.erb

1. <h1>Todos#index</h1>
2. <p>Find me in app/views/todos/index.html.erb</p>

Figure: todo app, index action screenshot

#### Displaying a todo from controller



Above steps are recap from [Module 1 - Lesson 4](#_bookmark11). Lets get onto some heavy- lifting now.

As first step, we will pass a todo item from controller to the view. Later, we will learn how to pass a bunch of todos. Even later, we will fetch the bunch from the database & show it on the browser.

The controller can pass variable @todo\_item1 to the view index.html.erb. Add the highlighted line in the file.

app/controllers/todos\_controller.rb

1. class TodosController < ApplicationController
2. def index
3. @todo\_item1 = "Buy Milk"
4. end

5. end

The instance variable @todo\_item1 can be used in the view to show the content passed by the controller.

Remove all the lines from the file index.html.erb & add the lines shown below.

app/views/todos/index.html.erb

1. ~~<h1>Todos#index</h1>~~
2. ~~<p>Find me in app/views/todos/index.html.erb</p>~~

3.

4. <title>Shared Todo App </title>

5. <h1>Shared Todo App</h1>

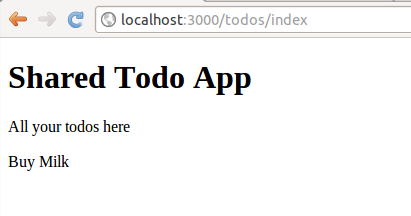
6. <p>All your todos here</p>

7. <p> <%= @todo\_item1 %> </p>

In the App Output, /todos/index should show as below.

Figure: Displaying a todo supplied by the controller

Instance variables, is the way the controller passes information to the view. Here @todo\_item is an instance variable.



The controller has access to the model (your database tables) and hence acts as the pipe between the model and the view. It gets data from the model and supplies it to view using instance variable.

<% ... %> is the way to embed Ruby code in the views. '<%=' prints the expression value to the view, while '<%' (without the '=' sign) simply evaluates the Ruby code & does not print anything.

Try replacing <p> <%= @todo\_item1 %> </p> with <p> <% @todo\_item1 %> </p> in the index.html.erb to see how /todos/index output changes.

Note Rails is not very hard. But what really stumps a first time learner is the huge number of files and all the magic that goes on. What you just saw was one of the rails magic, to have views access variables from the controller. You will see more as we move on. The more you understand this sort of magic, you will start to see how simple Rails actually is.

#### Little Ruby

Now is a good time to pick a bit of Ruby, as it would be needed going ahead.

Note When stuck with Ruby issues, you have two options. Either google for it like 'how to cocatenate strings in Ruby' or launch Rails console & start experimenting.

**Introducing Rails console**

Rails console can be started as below.

Console

user\_1@codelearn:~/shared\_todo\_app$ rails console Loading development environment (Rails 3.2.6)

1.9.3p194 :001 >

Rails console loads the complete Rails app environment. All the model & controller classes of the Rails app is available inside Rails console.

Getting back to the Ruby tutorial.

A comment in ruby starts with hash sign #.

You can print a string by print "hello world" OR puts "hello World".

For now the single and double quotes are same so - print 'hello world' OR puts 'hello World', would do as well.

Rails console

1.9.3p194 :001 > #This is a comment in ruby

1.9.3p194 :002 > print "Hello World" # Lets print Hello World Hello World => nil

1.9.3p194 :003 > puts "Hello World" Hello World

=> nil

1.9.3p194 :004 > print 'Hello World' #Single Quotes Hello World => nil

1.9.3p194 :005 > puts 'Hello World' Hello World

=> nil

Variables have no types and can store strings, integers etc, and everything is an object.

String concatenation is done with a +

Length of a string is calculated by calling .length function on a string object.

To know whether a string is empty use .empty?. Observe ? at the end of function name. Such functions return only true or false.

Rails console

1.9.3p194 :006 > first\_name = 'Hemanth'

=> "Hemanth"

1.9.3p194 :007 > last\_name = 'Haridas'

=> "Haridas"

1.9.3p194 :008 > first\_name + last\_name

=> "HemanthHaridas"

1.9.3p194 :009 > first\_name.length

=> 7

1.9.3p194 :010 > 'hello'.length

=> 5

1.9.3p194 :011 > first\_name.empty?

=> false

1.9.3p194 :012 > ''.empty?

=> true

Arrays in ruby are defined as arr = [ 21, 22, 23], the first element is arr[0] OR arr.first, second element is arr[1] , last element arr.last . Length of array

is arr.length

Accessing each element of array using blocks. arr.each { |a| puts a } prints all the elements of the array.

rails console

1.9.3p194 :001 > arr = [ 21, 22, 23]

=> [21, 22, 23] 1.9.3p194 :002 > arr[0]

=> 21

1.9.3p194 :003 > arr.first

=> 21

1.9.3p194 :004 > arr.last

=> 23

1.9.3p194 :005 > arr[1]

=> 22

1.9.3p194 :006 > arr.length

=> 3

1.9.3p194 :008 > arr.each { |a| puts a } 21

22

23

=> [21, 22, 23]

1.9.3p194 :009 > arr.each { |a| puts 2 \* a } 42

44

46

=> [21, 22, 23]

1.9.3p194 :010 >

Note Once done, you can exit Rails console by typing 'exit'

[← Previous](#_bookmark18) [Next →](#_bookmark24)

© Codelearn 2012