

|  |
| --- |
|  |
| Ruby On Rails  [Document subtitle] |
| |  |  |  | | --- | --- | --- | | geeksfarm | 11/13/19 | Ruby on rails | |

# Ruby Basic

Ruby is a dynamic, reflective, object-oriented, general-purpose programming language. It was designed and developed in the mid-1990s (first develop in 1995, first stable release 2014) by Yukihiro Matsumoto (aka "Matz") in Japan.

Writing Rules in Ruby

1. File name use this rule:
   * use lowercase.
   * use underline to connect two words.

Example:

* The right way:

user.rb

process\_data.rb

group7.rb

* The wrong way:

7user.rb

#$user.rb

Process.rb

1. Naming class, use this rule:
   * Use word in English or be consistent with your language.
   * Use singular for model class.
   * Use plural for controller class.
   * For class name use uppercase for first character in name class.

Example 1:

* Filename: user.rb
* Declared class:

class User

def initialize

end

end

Example 2:

* Filename:

calculate\_score*.rb*

* Declared class:

class CalculateScore

def initialize

end

end

1. For variable and action/function, use this rule:
   * use word in English or be consistent with your language.
   * use underline sign “\_” to connect two words.
   * use singular if only hold a value.
   * use plural if hold more than one values.
   * use lowercase for all words.

Example variable:

name\_student = “warner”

@name\_student\_2 = “Ujang”

Example method:

def calculate\_score

puts “your action here”

end

1. For indentation Use two space rather than tab. (or change tab to two space)

**String**

string in ruby is declared inside “” or '', the different between them is:

* Single quote will execute string exactly like the content inside it
* Double quote allows us to execute expression (like variable) and interpolation (like \n, \t) as is.

**Single quote:**

hero = 29

puts 'this is single quote \n and no #{hero}'

This will result:

this is single quote \n and no #{hero}

**Double quote:**

hero = 29

puts “this is double quote \n and no #{hero}”

This will result:

 this is double quote

and no 29

for complete source: [***http://en.wikibooks.org/wiki/Ruby\_Programming/Syntax/Literals#Strings***](http://en.wikibooks.org/wiki/Ruby_Programming/Syntax/Literals#Strings)

## Assignment

Assignment in ruby declare with equal sign '=', and there are many ways for assigning value to a variable.

**Self-Assignment**

name = “warner”

score = 3

score += 5

**Multiple Assignment**

name, score = “warner”, 3

name, score = [“warner”, 3]

**Conditional Assignment**

math = 32

score ||= math

Result will return math if score is nil, but if score not nil it will hold the current value.

math = 32

grammar = 88

score = math || grammar

Result will return math if grammar is nil, but if math nil and grammar not nil it will return grammar, and if math and grammar not nil it will return math.

## Conditional Statement

**if,**used for selection of specific condition, example:

if name.eql? "warner"

puts "that’s right the author is warner"

end

**elsif**, used for second selection of specific condition, example:

if name.eql? "warner"

puts "that right the author is warner"

elsif name.eql? "ujang"

puts "the editor is ujang"

end

**else**, used for last option if condition is not fulfilled, example:

if name.eql? "warner"

puts "that right the author is warner"

elsif name.eql? "ujang"

puts "the editor is ujang"

else

puts "didn't know who he is"

end

## Array

**Simple Array**

Simple array is variable that have index and can holds many values. This is how to create blank array:

scores = *Array*.new

And to insert values into an array, we can use '*<<'* or '*.push*', for example :

scores << 90

scores << 88

scores.push(70)

scores.push(69)

So if you want access data of array:

puts scores[0]

puts scores[1]

**Hash**

Hash is array that have custom key for each values. And this how to create blank hash:

Scores = *Hash*.new

And for inserting values to a hash, for example:

scores[“math”] = 90

scores[“grammar”] = 88

## Loop

loop use for repeated action in a sequence. In ruby many command that can do this action, but in this section we only describe three loop command.

**For**

Format:

for <variable> in <range\_data> do

puts “do something”

end

Example:

for n in 0..3 do

puts n

end

**While**

Format:

while condition do

puts “do something”

end

Example:

n = 0

while n < 3 do

puts n

n = n + 1

end

**Each**

Format:

<array>.each do |single\_object|

puts “do something”

end

Example:

scores = [9,8,7,10,11,8,5]

scores.each do |score|

puts score

end

## Variable Scope?

Scope defines where in a program a variable is accessible. Ruby has four types of variable scope, local, global, instance and class. In addition, Ruby has one constant type. Each variable type is declared by using a special character at the start of the variable name as outlined in the following table.

|  |  |
| --- | --- |
| **Name Begins with** | **Variable Scope** |
| $ | A global variable |
| @ | An instance variable |
| [a-z] or \_ | A local variable |
| [A-Z] A | constant |
| @@ | A class variable |

In addition, Ruby has two pseudo-variables which cannot be assigned values. These are nil which is assigned to uninitialized variables and self which refers to the currently executing object. In the remainder of this chapter we will look at each of these variable scopes in turn.

### Ruby Local Variables

Local variables are local to the code construct in which they are declared. For example, a local variable declared in a method or within a loop cannot be accessed outside of that loop or method. Local variable names must begin with either an underscore or a lower-case letter. For example:

loopcounter = 10

\_LoopCounter = 20

### Ruby Global Variables

Global variables in Ruby are accessible from anywhere in the Ruby program, regardless of where they are declared. Global variable names are prefixed with a dollar sign ($). For example:

$welcome = "Welcome to Ruby Essentials"

Use of global variables is strongly discouraged. The problem with global variables is that, not only are they visible anywhere in the code for a program, they can also be changed from anywhere in the application. This can make tracking bugs difficult.

It is useful to know, however, that a number of pre-defined global variables are available to you as a Ruby developer to obtain information about the Ruby environment. A brief summary of each of these variables is contained in the following table.

|  |  |
| --- | --- |
| **Variable Name** | **Variable Value** |
| $@ | The location of latest error |
| $\_ | The string last read by gets |
| $. | The line number last read by interpreter |
| $& | The string last matched by regexp |
| $~ | The last regexp match, as an array of subexpressions |
| $n | The nth subexpression in the last match (same as $~[n]) |
| $= | The case-insensitivity flag |
| $/ | The input record separator |
| $\ | The output record separator |
| $0 | The name of the ruby script file currently executing |
| $\* | The command line arguments used to invoke the script |
| $$ | The Ruby interpreter's process ID |
| $? | The exit status of last executed child process |

For example, we can execute the gets method to take input from the keyboard, and then reference the $\_ variable to retrieve the value entered:

irb(main):005:0> gets

hello

=> "hello\n"

irb(main):006:0> $\_

=> "hello\n"

Alternatively, we could find the process ID of the Ruby interpreter:

irb(main):007:0> $$

=> 17403

### Ruby Class Variables

A class variable is a variable that is shared amongst all instances of a class. This means that only one variable value exists for all objects instantiated from this class. This means that if one object instance changes the value of the variable, that new value will essentially change for all other object instances.

Another way of thinking of thinking of class variables is as global variables within the context of a single class.

Class variables are declared by prefixing the variable name with two @ characters (@@). Class variables must be initialized at creation time. For example:

@@total = 0

### Ruby Instance Variables

Instance variables are similar to Class variables except that their values are local to specific instances of an object. For example if a class contains an instance variable called @total, if one instance of the object changes the current value of @total the change is local to only the object that made the change. Other objects of the same class have their own local copies of the variable which are independent of changes made in any other objects.

Instance variables are declared in Ruby by prefixing the variable name with a single @ sign:

@total = 10

### Ruby Constant Scope

Ruby constants are values which, once assigned a value, should not be changed. I say should because Ruby differs from most programming languages in that it allows a constant value to be changed after it has been declared, although the interpreter will protest slightly with a warning message.

Constants declared within a class or module are available anywhere within the context of that class or module. Constants declared outside of a class or module are assigned global scope.

## Method

In this session will learn about type method, the first is class method then the second is instance method.

**Class method**

When a method is defined outside of the class definition, the method is marked as private by default. On the other hand, the methods defined in the class definition are marked as public by default. The default visibility and the private mark of the methods can be changed by public or private of the Module.

Whenever you want to access a method of a class, you first need to instantiate the class. Then, using the object, you can access any member of the class.

Only can access from object class itself, example:

class Student

def self.name

puts “Hello Brother”

end

end

*Student*.name

This will result:

    Hello Brother

* *self*, use for describing if action/method name is part of class itself.

**Instance Method**

only can access when object class has become instance, example:

class Student

def name

puts “Hello Brother”

end

end

student = *Student*.new

student.name

This will result:

    Hello Brother

## Class

In ruby, class can be created like any other programming language, but the different is when we need to use properties (setter and getter).

This is an example when we use common style:

class Student

# initialize is constructor for class

def initialize

@name = "hero"

@grade = 4

end

def get\_name

@name

end

def get\_grade

@grade

end

def set\_name *name*

@name = name

end

def set\_grade *grade*

@grade = grade

end

end

The initialize method is a standard Ruby class method and works almost same way as constructor works in other object oriented programming languages. The initialize method is useful when you want to initialize some class variables at the time of object creation. This method may take a list of parameters and like any other ruby method it would be preceded by def keyword.

So when you want create new object class:

student = *Student*.new

#if you want get name

student.get\_name

#if you want change name

student.set\_name "hero our time"

Above class code is the old way when you want access variable object in a class, you must set manually setter and getter, but ruby have action helper for handle that both, first is:

* *attr\_writer*, this is use for create setter.
* *attr\_reader*, this is use for create getter.
* *attr\_accessor*, this is use for create setter and getter.

This is how you create class in ruby way:

class Student

attr\_accessor :name, :grade

# initialize is constructor for class

def initialize

@name = "hero"

@grade = 4

end

end

So if you want to create new object class:

Student = *Student*.new

#if you want get name

student.name

#if you want change name

student.name = "hero our time"

That will do the same with the first example, but if you notice in ruby way you don't need manually declare properties for a class, ruby has function for creating properties the simple way.

There may be a situation when you want to create an object without calling its constructor initialize i.e. using new method, in such case you can call allocate, which will create an uninitialized object for you as in the following example:

#using allocate

class Box

attr\_accessor :width, :height

# constructor method

def initialize(*w*,*h*)

@width, @height = w, h

end

# instance method

def getArea

@width \* @height

end

end

# create an object using new

box1 = *Box*.new(10, 20)

# create another object using allocate

box2 = *Box*.allocate

# call instance method using box1

a = box1.getArea()

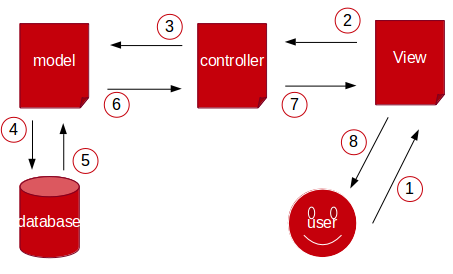
puts "Area of the box is : #{a}"

# call instance method using box2

a = box2.getArea()

Complete Resources: [***http://ruby-doc.org/core-2.0.0/***](http://ruby-doc.org/core-2.0.0/)

# MVC Work-Flow in Ruby On Rails



**Description:**

1. User access a web-page and click a button to show all data.
2. View page calling method from controller.
3. Controller access query via model.
4. Model will execute query to get all data.
5. Database will give data that requested by model.
6. Model will get all data from database, and send them to the controller.
7. Controller will get all data from model and put that in a variable that will used by view.
8. View will show the data to a web-page so user can see the result.

# Create Application

To create an application, make sure your computer connects to the internet, open terminal and execute:

rails new <name\_application> -d=postgresql

Wait for several minutes because it will take a time for it to complete. If you’re using mysql use -d=mysql.

'-d=postgresql' use to determine, which database that will be used by the application, in this example is Postgresql, or we can use ‘-d=mysql’.

If app fails to install bundle, you can manually install it, open terminal:

cd <name\_application\_folder>

bundle install

That will install all gem via internet connection, or if you want install locally use this:

bundle install --local

But if you use local installation make sure you have install all gem that use by Gemfile in your local machine.

\*not recommended install gem locally

if you check to rails project directory this structure that normally created:

**app/** Contains the controllers, models, views, helpers, mailers and assets for your application.

**bin/** Contains the rails script that starts your app and can contain other scripts you use to deploy or run your application.

**config/** Configure your application's runtime rules, routes, database, and more.

**config.ru** Rack configuration for Rack based servers used to start the application.

**db/** Contains your current database schema, as well as the database migrations.

**Gemfile** These file use for determine what gem is used by application.

**Gemfile.lock** Gem dependencies are needed for your Rails application. Used by gem Bundler.

**lib/** Extended modules for your application.

**log/** Application log files.

**public/** The only folder seen to the world as-is. Contains the static files and compiled assets.

**Rakefile** This file locates and loads tasks that can be run from the command line. For custom rake task you need adding files to the lib/tasks directory of your application.

**README.rdoc** This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on.

**test/** Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications

**tmp/** Temporary files (like cache, pid and session files)

**vendor/** A place for all third-party code. In a typical Rails application, this includes Ruby Gems and the Rails source code (if you optionally install it into your project).

## Troubleshoot

**- Error installing pg. probably of neccessary library**, that means in your ubuntu machine is lack of library of postgres for development, for handle this error we need install *libpq-dev* in our local, on terminal run:

sudo apt-get install libpq-dev

***ExecJS::RuntimeUnavailable****,*that means in your Ubuntu machine is lack of library nodejs, for handle this error we need install the *nodejs* in Ubuntu machine, on terminal run:

sudo apt-get install nodejs

# Database

Database configuration resides in <name\_project\_folder>/config/database.yml, open and change the username and password to match with your local database setting:

default: &default

adapter: mysql2

encoding: utf8

pool: 5

username: root

password: geeks

socket: /var/run/mysqld/mysqld.sock

development:

<<: \*default

database: coba\_ror\_development

test:

<<: \*default

database: coba\_ror\_test

production:

<<: \*default

database: coba\_ror\_production

username: coba\_ror

password: <%= ENV['COBA\_ROR\_DATABASE\_PASSWORD'] %>

After configuring database.yml file you need to create database, open terminal and type:

rake db:create

or

rails db:create

“Since rails 5 all command incorporated into rails so we can use rake or the newest one rails”.

If you check your database, there will be new blank database development and test, it still doesn’t have table.

To check if the application successfully created, you can try run rails server, open console:

cd <your\_project\_directory>

rails s

And in console you should see this (example):

=> Booting WEBrick

=> Rails 4.1.4 application starting in development on http://0.0.0.0:3000

=> Run `rails server -h` for more startup options

=> Notice: server is listening on all interfaces (0.0.0.0). Consider using 127.0.0.1 (--binding option)

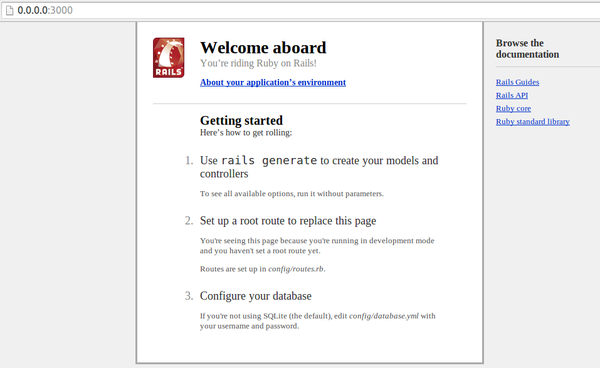
=> Ctrl-C to shutdown server

[2014-09-09 13:37:41] INFO WEBrick 1.3.1

[2014-09-09 13:37:41] INFO ruby 2.1.2 (2014-05-08) [x86\_64-linux]

[2014-09-09 13:37:41] INFO WEBrick::HTTPServer#start: pid=10285 port=3000

that means you rails server successfully running, in second line you will get link to your application http://0.0.0.0:3000, open your browser and in address bar put that link, you should get this page:



“make sure you’re on your root project directory if you want run: rails server, rails console, rails generate, rake”

## Troubleshoot

If an error occurred when installing gem 'pg' or 'mysql2' ("Failed to build gem native extension"), that means your local pc need install library for development:

For gem 'pg', in terminal run:

sudo apt-get install libpq-dev

For gem 'mysql2’, in terminal run:

sudo apt-get install mysql-client libmysqlclient-dev

# Model

Before we create model we need table 'schema\_migration', open the terminal and execute:

rails db:migrate

That table used for log all file migration that has been executed so when we migrate again it won’t execute again. This table only contain column 'version' with type varchar.

Now we will create model article, open terminal type:

rails g model article title:string content:string --no-test-framework

It will generate:

invoke active\_record

create db/migrate/20140909070550\_create\_articles.rb

create app/models/article.rb

if you open article.rb, this code inside the file:

class Article < ActiveRecord::Base

end

In rails 3 if you want to access database column, you have to setup attr\_accessible for every columns name (for “mass assignment” = insert, delete, edit a data or multiple data), but in rails 4 or 5 it won’t work, you must manually call model and column name in controller. (this will describe in the discussion of the controller).

And if you open 20140909070550\_create\_articles.rb, this code inside the file:

class CreateArticles < *ActiveRecord::Migration*

def change

create\_table :articles do |t|

t.string :title

t.string :content

t.timestamps null: false

end

end

end

In migration file you can add or remove column. You have created model but table articles still not created. It’s because when we generate model, rails engine only creates class model and migration for table (this file that will execute when call command rake). To create table, open terminal type:

rails db:migrate

That command will execute file in folder db/migrate/\*, and if you check database there must be table 'articles'. And if you check table 'schema\_migration' there must record '20140909070550'.

\*--skip-test and –no-test-framework, that command will skip model generator for create file unit test for model that created.

\*if you want delete file model don't use manual way, try use rails destroy command, example:

rails g model article --skip-test --no-test-framework

that will create:

create db/migrate/20140909070550\_create\_articles.rb

create app/models/article.rb

Then if you want to delete files that has been generated by that model, use this command:

rails d model article

Delete all file that has been created by the model generator. This what you must get from console:

remove db/migrate/20140909035814\_create\_articles.rb

remove app/models/article.rb

## Table Manipulation

If you want manipulate the table that has been created, you can use rails migration generator command:

rails g migration <description\_action\_file>

this is how to use it, on terminal:

rails g migration add\_status\_to\_articles

That will generate:

invoke active\_record

create db/migrate/20140909072020\_add\_status\_to\_articles.rb

if you open 20140909072020\_add\_status\_to\_articles.rb file you will get this code:

class AddStatusToArticles < *ActiveRecord::Migration*

def change

end

end

That code still doesn’t have action, so you need to add an action for modifying the existing table, this several functions that use by rails to manipulate table:

1. Add column

Add new column to table:

add\_column :<name\_table>, :<name\_column>, :<type\_column>

and this how to use it:

class AddStatusToArticles < *ActiveRecord::Migration*

def change

add\_column :articles, :status, :string

add\_column :articles, :status\_two, :string

end

end

1. Remove column

Delete column from table:

remove\_column :<name\_table>, :<name\_column>

and this how to use it:

class RemoveStatusFromArticles < *ActiveRecord::Migration*

def change

remove\_column :articles, :status

end

end

1. Rename column

Rename existing column from table:

rename\_column :<name\_table>, :<old\_column\_name>, :<new\_column\_name>

and this how to use it:

class RenameStatusToStatusTwoFromArticles < *ActiveRecord::Migration*

def change

rename\_column :articles, :status, :status\_two

end

end

1. Change column

change type without dropping existing column:

change\_column :<name\_table>, :<name\_column>, :<new\_type\_column>

and this how to use it:

class ChangeStatusFromArticles < *ActiveRecord::Migration*

def change

change\_column :articles, :status, :integer

end

end

1. Rename table

rename table name:

rename\_table :<old\_table\_name>, :<new\_table\_name>

and this how to use it:

class RenameArticlesToNews < *ActiveRecord::Migration*

def change

rename\_table :articles, :news

end

end

1. Drop table

delete existing table:

drop\_table :<table\_name>

and this how to use it:

class DeleteNews < *ActiveRecord::Migration*

def change

drop\_table :news

end

end

1. Create table

create new table:

create\_table :<table\_name> do |type|

type.<type\_of\_column> :<name\_column>

type.<type\_of\_column> :<name\_column>

end

and this how to use it:

class CreateArticles < *ActiveRecord::Migration*

create\_table :articles do |t|

t.string :title

t.string :content

t.timestamps

end

end

\*not recommended create table via migration, use 'model generator' for create table.

After modifying migration file, don't forget to run:

rails db:migrate

\*make sure you restart rails server or rails console after:

add new migration file, add new model, modify model, routes file.

## Migration Manipulation

You have already using rails command that generate migration for creating table or table manipulation, this is the description about rake or rails command that execute migration file:

1. Execute File Migration

this rake command will execute file migration, but if the code of migration file has been saved to table 'schema\_migration' that file won’t be executed.

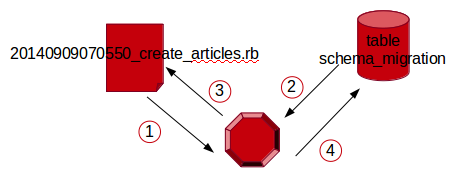
rails db:migrate

For example, if you have file:

- 20140909070550\_create\_articles.rb

- 20140909072020\_add\_status\_to\_articles.rb

then you run 'rake db:migrate', this is what happens:



Case where version number not exist in schema\_migration:

1. rails db:migrate will get version number 20140909070550.
2. then search in table schema\_migration.
3. if version number not present it will execute file migration.
4. if success rails db:migrate will insert version number into table schema\_migration.

Case where version number present in schema\_migration :

1. rails db:migrate will get version number 20140909070550.
2. then search in table schema\_migration.
3. if version number present it will send messages, that inform if version number has present.

\*if you execute migration then your code error, the version number will not save in schema\_migration.

\*And careful if you delete manually file migration but you have executed that file, that will cause problem, you need to delete data from schema\_migration table that hold version number file that you deleted.

1. Rollback Migration

this rails command will rollback file migration, and remove data version number from table 'schema\_migration'.

rails db:rollback

That will reverse last migration that has been executed.

If you want rollback to specific level use:

rails db:rollback STEP=2

will reverse two last migrations that has been executed.

1. Restart Migration

if you modify file migration and want to reload migration (do rollback then call migrate), use:

rails db:migrate:redo

That will reverse last migration and migrate again after that.

If you want to restart migration specific level use:

rails db:migrate:redo STEP=2

Will reverse two last migrations then migrate again from two last migrations.

1. Setup Database

For new development to setup the database with:

rails db:setup

that command will create database and run schema, seed data.

1. Restart Database

If you want to delete and re-create database and fill with seed data, you can use:

rails db:reset

that will drop database and re-create database then run the schema and seed data.

1. Seeding Database

Seeding usually used for creating test data so we don’t have to insert data one by one manually. To do this we can use.

rails db:seed

1. Drop Database

If you want to delete database, you can use:

rails db:drop

1. Dump Schema Table

If you have existing database, and you want to create schema.rb file from existing tables, you can use:

rails db:schema:dump

after run that code check you project\_name/db folder there should be create schema.rb file.

1. Load Schema Table

If you have existing schema.rb file, and you want to create table from structure that file, you can use:

rails db:schema:load

1. Dump Structure Table

If you have existing table, and you wanna create sturcture.sql from that tables, you can use:

rails db:structure:dump

1. Load Structure Table

If you have existing structure.sql file, and you want create tables from those file, you can use:

rails db:structure:load

More Resources:

[http://stackoverflow.com/questions/10301794/difference-between-rake-dbmigrate-dbreset-and-dbschemaload](http://stackoverflow.com/questions/10301794/difference-between-rake-dbmigrate-dbreset-and-dbschemaload%20)

## Active Record

Active Record is a layer for handling business data and logical manipulation. If you ever doing query with SQL syntax, it is the same in active record just in different way. This is the comparison between SQL syntax and Active Record:

1. **Insert Data**

SQL: insert into <name\_table> (<name\_column>) values <values>

example:

insert into articles(title, content) values("Sangkuriang", "di suatu hari sangkuriang murka dan menedang perahu hingga berubah jadi tangkuban parahu");

Active Record: <ModelName>.create(<name\_column>: <values>)

example:

*Article*.create(title: "Kabayan", content: "di rumah pak rw terjadi keramaian karena kabayan ngamuk ingin ketemu ama cepot")

Or:

article\_new = *Article*.new

article\_new.title = "Semar Astana Nyar"

article\_new.content = "kelaparan - kelaparan semar berteriak sambil berharap ada yang memberi makan"

article\_new.save

1. **Select Data**

Get all data from table

SQL: select <option> from <name\_table>

example:

Select \* from articles;

Active Record : <ModelName>.all

example:

*Article*.all

Get specific column

SQL: select <option\_name\_column> from <name\_table>

example:

Select title from articles;

Active Record : <ModelName>.<option>.select(<option\_name\_column>)

example:

*Article*.select(:title)

Get a single data

SQL: select <option> from <name\_table> where <condition>

example:

Select \* from articles where id = 2;

Active Record: <ModelName>.find(<id>)

example:

*Article*.where(id: 2)

\*This is bad example to get a record, never use above query if want get one data, use:

*Article*.find(2)

\*use this query if you want get a record by id, but when data not found it will return error

Or: <name\_model>.find\_by\_<name\_column>(<values>)

*Article*.find\_by\_id(2)

\*use this query if you want get one data by name of column, if not found it will return nil. (use this instead)

Get data with sorting

SQL: select <option> from <name\_table> order by <name\_column> <asc/desc>

example:

Select \* from articles order by id desc;

Active Record: <ModelName>.<option>.order(“<name\_column> <asc/desc>”)

example:

*Article*.order(“id desc”)

1. **Edit Data**

Edit One Record

SQL: update from <name\_table> set <options> where <condition>

example:

update articles set title = "Tangkuban Parahu" where id = 1;

Active Record: <ModelName>.<option>.update(<set\_value\_to\_specific\_column>)

example:

*Article*.find\_by\_id(1).update(title: “Semar Astanya”)

Or:

article\_edit = *Article*.find\_by\_id(1)

article\_edit.update(title: “Rama Yana”)

Or:

article\_edit = *Article*.find\_by\_id(1)

article\_edit.title = “Rama Yana”

article.save

Edit Multiple Data

SQL: update from <name\_table> set <options>

example:

update articles set status = "true";

Active Record: <ModelName>.update\_all(<set\_value\_to\_specific\_column>)

example:

*Article*.update\_all(status: “active”)

1. **Delete Data**

Delete One Data

SQL: delete from <name\_table> where <condition>

example:

Delete from articles where id = 1;

Active Record: <ModelName>.<option>.destroy

example:

*Article*.find\_by\_id(1).destroy

Delete All Data

SQL: delete from <name\_table>

example:

Delete from articles;

Active Record: <ModelName>.delete\_all

example:

*Article*.delete\_all

Complete Resources: <http://guides.rubyonrails.org/active_record_querying.html>

# Controller

## Routes

The routes in rails is used for managing URL in application, so when you access a link in browser it will recognizes and process it. This is example of restful routes in rails (Representational state transfer [REST] is an abstraction of the architecture of the World Wide Web).

**HTTP Verb     URL                Controller         Action         Used for**

GET              /articles             Articles              index             display all article in an overview

GET              /articles/new     Articles               new              return an HTML form for creating a new article

POST            /articles             Articles              create           create a new article

GET              /articles/1          Articles              show             display a specific article

GET              /articles/1/edit    Articles              edit               return an HTML form for editing an article

PUT              /articles/1           Articles              update          update a specific article

DELETE       /articles/1/          Articles              destroy         delete a specific article

That is complete routes if you want use manipulate data for specific table.

Implementation

In RoR project you must setup routes in config/routes.rb, in here you can declare the group of routes that you will use.

### Base Route

These routes are to declare base routes, usually used by home page of a website.

root “<controller\_name>#<action\_name>”

Example:

*Rails*.application.routes.draw do

root "articles#index"

end

Try run ‘rails routes’ in console you will get:

Prefix Verb URI Pattern Controller/Action

root GET / articles#index

### Restful Route

this routes for create resourceful routes so it will automatic setup CRUD path.

resources :<name\_controller>

Example:

*Rails*.application.routes.draw do

resources :bicycles

end

Try run '***rails routes***' in console you will get:

**Prefix Verb URI Pattern Controller/Action**

bicycles GET /bicycles(.:format) bicycles#index

POST /bicycles(.:format) bicycles#create

new\_bicycle GET /bicycles/new(.:format) bicycles#new

edit\_bicycle GET /bicycles/:id/edit(.:format) bicycles#edit

bicycle GET /bicycles/:id(.:format) bicycles#show

PATCH /bicycles/:id(.:format) bicycles#update

PUT /bicycles/:id(.:format) bicycles#update

DELETE /bicycles/:id(.:format) bicycles#destroy

### Custom Setup Route

this routes are used to create manual routes so it will create custom route.

<verb\_method> '<uri\_pattern>' => '<controller\_name>#<action\_name>', as: :<name\_prefix>

Example:

*Rails*.application.routes.draw do

get 'bicycles/:id/buy' => 'bicycles#buy', as: :buy

end

Try run '***rails routes***' in console you will get:

**Prefix Verb URI Pattern Controller/Action**

buy GET /bicycles/:id/buy bicycles#buy

### Grouping Route with Member and Collection

#### Member

this route is to create group of routes and when user access an URL, it will need an input.

resources :<name\_controller> do

member do

<verb\_method> '<name\_action>'

end

end

Example:

*Rails*.application.routes.draw do

resources :bicycles do

member do

get 'special'

post 'discount'

end

end

end

Try run '***rails routes***' in console you will get:

**Prefix Verb URI Pattern Controller/Action**

special\_product GET /bicycles/:id/special(.:format) bicycles#special

discount\_product POST /bicycles/:id/discount(.:format) bicycles#discount

#### Collection

This route used to create group routes and when user access URL it doesn’t need an input.

resources :<name\_controller> do

collection do

<verb\_method> '<name\_action>'

end

end

Example:

*Rails*.application.routes.draw do

resources :bicycles do

collection do

get 'special'

post 'discount'

end

end

end

Try run '***rails routes***' in console you will get:

**Prefix Verb URI Pattern Controller/Action**

special\_products GET /bicycles/special(.:format) bicycles#special

discount\_products POST /bicycles/discount(.:format) bicycles#discount

### Namespace Route

This route used to create namespace routes so it will create folder structure when accessing controller file.

namespace :<name\_folder> do

resources :<name\_controller>

end

Example:

*Rails*.application.routes.draw do

namespace :api do

resources :bicycles

end

end

Try run '***rails routes***' in console you will get:

**Prefix Verb URI Pattern Controller/Action**

api\_bicycles GET /api/bicycles(.:format) api/bicycles#index

POST /api/bicycles(.:format) api/bicycles#create

new\_api\_bicycle GET /api/bicycles/new(.:format) api/bicycles#new

edit\_api\_bicycle GET /api/bicycles/:id/edit(.:format) api/bicycles#edit

## Create Controller

Let’s create controller for articles, this controller will become bridge between model and view, so when you want to show list of data in view then controller will request from model.

Open terminal and type:

rails g controller articles index new edit --no-helper --no-assets --no-test-framework

That will generate:

create app/controllers/articles\_controller.rb

route get 'articles/edit'

route get 'articles/new'

route get 'articles/index'

invoke erb

create app/views/articles

create app/views/articles/index.html.erb

create app/views/articles/new.html.erb

create app/views/articles/edit.html.erb

when you generate controller, it will create:

1. articles\_controller.rb, this file is where you put code for handle articles. If you open this file code will look like:

class ArticlesController < *ApplicationController*

def index

end

def new

end

def edit

end

end

the function index, new, edit automatically created. This function use for handling actions for view index, new, & edit.

1. Set routes in **name\_app/config/routes.rb**, routes are address/path that used by view to access controller. Open the file:

*Rails*.application.routes.draw do

get 'articles/index'

get 'articles/new'

get 'articles/edit'

end

If you notice there are code *get 'articles/index',*this means if view access link *articles/index* it will use method *get*. And you can check URL that ready to be used, open terminal type:

rails routes

And it will result:

**Prefix                  Verb          URI Pattern                       Controller#Action**

articles\_index      GET          /articles/index(.:format)         articles#index

articles\_new        GET          /articles/new(.:format)           articles#new

articles\_edit         GET         /articles/edit(.:format)            articles#edit

That means your route for accessing article index, new, edit is ready. But it’s not a complete restful route yet, if you want restful, modify your *routes.rb* file into:

*Rails*.application.routes.draw do

root "articles#index"

resources :articles

end

This if you check with *rails routes:*

**Prefix           Verb          URI Pattern                   Controller#Action**

root              GET          /                                       articles#index

articles         GET          /articles(.:format)               articles#index

                    POST       /articles(.:format)               articles#create

new\_article   GET          /articles/new(.:format)        articles#new

edit\_article    GET         /articles/:id/edit(.:format)     articles#edit

article           GET         /articles/:id(.:format)           articles#show

                    PATCH    /articles/:id(.:format)            articles#update

                    PUT        /articles/:id(.:format)             articles#update

                    DELETE  /articles/:id(.:format)             articles#destroy

root, is default path, example if you request [*http://0.0.0.0:3000*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2F&sa=D&sntz=1&usg=AFQjCNGmODCaaoGT5HHOms_EZ8eAwZuAnA)you will get page from [*http://0.0.0.0:3000/articles*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles&sa=D&sntz=1&usg=AFQjCNHXKddKArAeZ-aDS3_PeAmI8YcnTA).

1. Createview**index.html.erb, new.html.erb, edit.html.erb**, this file is where to put html (inside body tag) and data from database that will show on web-page. If you open index.html.erb, the code will look like:

<h1>Articles#index</h1>

<p>Find me in app/views/articles/index.html.erb</p>

Index.html.erb is loaded by layout, *app/view/layouts/****application.html.erb****,*this file is used for loading css, javascript, and all view tagif you open the application.html.erb file, the code will look like:

<!DOCTYPE html>

<html>

<head>

<title>DummyRail5</title>

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

<%= javascript\_include\_tag 'application', 'data-turbolinks-track' => true %>

<%= csrf\_meta\_tags %>

</head>

<body>

<%= yield %>

</body>

</html>

If you notice code above there’s a *data-turbolinks-track,* this code used for tracking certain assets, so when you access page asset it’s always the latest version inside turbolinks (turbolinks is gem that use for make following link in web application become faster, and only working if action is *get*) session.

Then there is *yield* that use to show main content from all view, for example if you have generated view: *app/views/articles/index.html.erb,* this view will show by *yield*from *layout.html.erb.*

Now try restart the rails server, then in browser open *http://0.0.0.0:3000/articles/.*

Now add file *show.html.erb*(will use at CRUD section) manually (**if you didn’t generate it automatically while creating controller**) to *app/views/articles/,*so view structure looks like:

- app/

- views/

---- index.html.erb

---- new.html.erb

---- edit.html.erb

---- show.html.erb

then for action update and destroy we don't need the view because method in controller will redirect or render specific page after success or fails to do the action.

## Views in Rails

The View layer is responsible for generating the HTML response for the request. Ruby on Rails is built upon the principles of the MVC paradigm. This makes it really easy to create an MVC application.

By convention, Rails will automatically route Controller methods to specifically named Views.

For example, add the following route to your routes.rb file:

get "articles", to: "articles#index", as: :articles

Next add the following Controller:

class ArticlesController < *ApplicationController*

def index

end

end

Notice we aren’t calling the View in the index method.

Now if you fire up your Rails application and visit /articles in the browser you should see Rails complaining that the View file does not exist.

Given this scenario, Rails will expect that there should be a View file under views/articles called index.html.erb.

If you add that file and then refresh the application, you should see the page is displayed correctly.

As with all of the other conventions of Ruby on Rails, this is designed to make your life easier, but it can be handy to understand the conventions so you know what is going on under the hood.

### The render method

As you can see from the example above, we don’t need to explicitly render the View because Rails will **assume** that’s what we want to do.

**But sometimes you will need to tell Rails to render a different View**.

For example, when you are updating a record, but the user has caused a validation error, you need to send the user back to the edit form so they can correct their mistake:

def update

@article = *Article*.find(params[:id])

if @article.update(article\_params)

redirect\_to(@article)

else

render :edit

end

end

In the example above, when the update method returns false I’m calling the render method with a symbol of :edit. This will cause the edit.html.erb template to be rendered.

### Working with layouts

If you look under the layouts directory, under the views directory, you should find a file called application.html.erb.

This is the default layout file that will be used to render the responses.

When choosing the layout to render, Rails will first look to see if there is a Controller specific layout.

For example, if we wanted to display a different layout for the ArticlesController methods, we could create a articles.html.erb file under the layouts directory.

**You can also specify the layout you want to use in the Controller:**

class ArticlesController < *ApplicationController*

layout "content"

end

### Yielding the content

As you can see from the default application.html.erb file, there is the following tag:

<%= yield %>

The yield tag is where the content from the View will be injected.

In the default example, the application.html.erb file contains the header and the footer of the website, and the content is then injected into this template via the call to yield.

You can also have multiple yield areas in a layout, for example:

<div class="main">

<%= yield %>

</div>

<div class="sidebar">

<%= yield :sidebar %>

</div>

Here we’ve got the default yield, but also a named :sidebar area.

Now in your View file you can specify the content for the sidebar:

<% content\_for :sidebar do %>

<p>I’m in the sidebar</p>

<% end %>

<p>Hello, World!</p>

### Partial Layouts

A good practice when building up the View layout of a web application is to extract components into Partials.

This makes it easy to deal with certain aspects of the design in isolation, and makes it easier to use the same components in multiple places throughout the application.

Here’s how you render a partial:

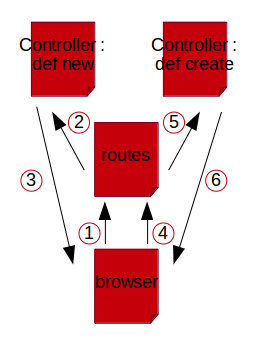
<%= render "sidebar" %>

This will render the file \_sidebar.html.erb. By default, partials should be named with a preceding underscore to show that they are partials. However, you don’t need to use the underscore in the name when rendering a partial.

## Implement CRUD in view and controller

### Create new data

Work-flow Create new data:



**Description:**

1. Browser request to [*http://0.0.0.0:3000/articles/new*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Fnew&sa=D&sntz=1&usg=AFQjCNHrjPZBaN8YFdMM2TY_ivPPk1GQow)*.*
2. Routes will check if that path registered, if yes it will find controller *articles* and action *new*.
3. Then will render view for *new* on browser, this page contains input form for new data.
4. User click submit button it will send data via [*http://0.0.0.0:3000/articles*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles&sa=D&sntz=1&usg=AFQjCNHXKddKArAeZ-aDS3_PeAmI8YcnTA) with method *post*.
5. Routes will check if path for *create*is registered, if yes it will find controller *articles* and action *create,*then process the parameter to be saved in table *articles.*
6. If success it will redirect to page index but if fails it will render page new.

Create new data to table is need form input and process in controller. First we will modify method new, open article controller file, lets open it and change into:

def new

@article = *Article*.new

end

Then now we will create form in view, open new.html.erb, and modify content into:

<%= form\_for @article do |f| %>

<%= f.label :title %>

<%= f.text\_field :title %>

<br/>

<%= f.label :content %>

<%= f.text\_area :content, cols: 20, rows: 10 %>

<br/>

<%= f.label :status %>

<%= f.radio\_button :status, "active" %>

<%= label :article\_status, 'active' %>

<%= f.radio\_button :status, "off", :checked => true %>

<%= label :article\_status, 'off' %>

<br/>

<%= f.submit :submit %>

<% end %>

Then run rails server, and on browser open [*http://0.0.0.0:3000/articles/new*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2Fnew&sa=D&sntz=1&usg=AFQjCNFZ3ces90KQr_ysozi9Su0WvUSSig).

If you try to submit it will cause error missing template, that because you still not setup method create, open article controller then add code:

def create

@article = *Article*.new(params[:article])

@article.save

flash[:notice] = "Success Add Records"

redirect\_to action: 'index'

end

Save that code and try again submit form, then you will get error *ForbiddenAttributesError,*that because in rails 4 parameter treated as strong parameter, that means if you want access specific parameter from controller you must add it to the white-list. This is how you use parameter in rails:

def create

@article = *Article*.new(params\_article)

@article.save

flash[:notice] = "Success Add Records"

redirect\_to action: 'index'

end

private

def params\_article

params.require(:article).permit(:title, :content, :status)

end

*- params.require(:article)*will get parameter article, inside this parameter have child title, content, status.

- .permit(:title, :content, :status) will let parameter title, content, status that processed for save data into database.

Now try to submit again, the success case is data must be saved to table article. The problem is when you submit empty data it will saved too, that means you need validation when data is being submit. In Ruby on Rails data validation will be put in model, let’s open model article (*article.rb*), change it into this:

class Article < *ActiveRecord::Base*

validates :title, presence: true,

length: { minimum: 5 }

validates :content, presence: true,

length: { minimum: 10 }

validates :status, presence: true

end

Validation is added for all field that require filled and forbidden blank, and for title, content is to add minimum character that can be submit. If you try insert data that not valid it will stay in the same page, but you don't know which input field that error. Now we will add error message base on each input, open article new view (*new.html.erb*), modify code into:

<%= form\_for @article do |f| %>

<%= f.label :title %>

<%= f.text\_field :title %>

<%= @article.errors[:title].first unless @article.errors[:title].blank? %>

<br/>

<%= f.label :content %>

<%= f.text\_area :content, cols: 20, rows: 10 %>

<%= @article.errors[:content].first unless @article.errors[:content].blank? %>

<br/>

<%= f.label :status %>

<%= f.radio\_button :status, "active" %>

<%= label :article\_status, 'active' %>

<%= f.radio\_button :status, "off", :checked => true %>

<%= label :article\_status, 'off' %>

<br/>

<%= f.submit :submit %>

<% end %>

- @article.errors, will return has array error of article base on rule validation in model.

- .first, is method helper that provide by rails, that used for get first record from array.

Try submit blank data.

For complete documentation can be found here:

[*https://guides.rubyonrails.org/v5.2/active\_record\_validations.html*](https://guides.rubyonrails.org/v5.2/active_record_validations.html)

Now open article controller modify method create into this:

def create

@article = *Article*.new(params\_article)

if @article.save

flash[:notice] = "Success Add Records"

redirect\_to action: 'index'

else

flash[:error] = "data not valid"

render 'new'

end

end

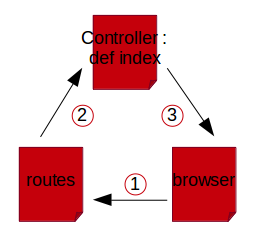
In that example we will check if article success saved or not, when success saved it will redirect page into index, but if data not valid it will render same page and send flash error message. Different between render and redirect is data hold or not in browser when web page is refreshed.

For complete documentation can be found here:

[*http://stackoverflow.com/questions/7493767/are-redirect-to-and-render-exchangeable*](http://stackoverflow.com/questions/7493767/are-redirect-to-and-render-exchangeable)

### List record

Work-flow List Record:



**Description:**

1. Browser Request [*Http://0.0.0.0:3000/Articles*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles&sa=D&sntz=1&usg=AFQjCNHXKddKArAeZ-aDS3_PeAmI8YcnTA).
2. Routes Will Check If That Path Exist, If Yes Routes Will Find Controller *Articles* and Action *Index.*
3. Then action *index*will find all data, then show all list data to index view that will show in browser.

List data will be showed in *index.html.erb*, and the function *index* from *articles\_controller.rb*that will handle query for select all data. Let’s open articles controller file, and modify def index, into:

def index

@articles = *Article*.all

end

Now we will use global variable @articles from index.html.erb, open file view index, and modify content into this file:

<% @articles.each do |article| %>

<h3><%= article.title %></h3>

<p><%= article.content %></p>

<h4>status : <%= article.status %></h4>

<hr/>

<% end %>

Now when you access [*http://0.0.0.0:3000/articles/*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2F&sa=D&sntz=1&usg=AFQjCNEKqS3UNEt-ukPUgvAqmLl-KArCwA)you will get list of data.

But if you notice status, boot article that have status active or off is showing, the right case is when article that status active that only showing, if off is not showing. To handle this, we will be using scope (scope is method that handle query so you don't need to repeat writing same query, but you just call name scope), scope dived into two, custom scope and default scope:

- **Custom Scope**

In article model (article.rb):

scope :status\_active, -> {where(status: 'active')}

In controller article (articles\_controller.rb):

def index

@articles = *Article*.status\_active

end

Now if you access index page the article with status active that will show in the list.

- **Default Scope**

in article model (article.rb):

default\_scope {where(status: 'active')}

in controller article (articles\_controller.rb):

def index

@articles = *Article*.all

end

The different between custom scope and default scope is when we call query the scope automatically added where condition from default\_scope in every query that access article model, this mean when you do select, create, update, delete in model article the default scope will run.

For complete documentation can be found here:

[*http://edgeguides.rubyonrails.org/active\_record\_querying.html#scopes*](http://edgeguides.rubyonrails.org/active_record_querying.html%23scopes)

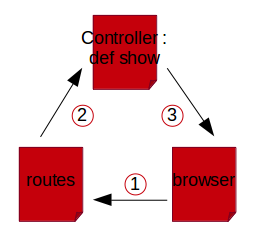
[*http://stackoverflow.com/questions/20517318/rails4-scope-with-parameters*](http://stackoverflow.com/questions/20517318/rails4-scope-with-parameters)

[*http://stackoverflow.com/questions/25491903/rails-scope-multiple-params*](http://stackoverflow.com/questions/25491903/rails-scope-multiple-params)

[*http://stackoverflow.com/questions/20597139/difference-lambda-scope-class-method-in-rails-best-practice*](http://stackoverflow.com/questions/20597139/difference-lambda-scope-class-method-in-rails-best-practice)

### Show data

**Work-flow Show Data:**



**Description:**

1. Browser click edit link in list data, the data have id = 1, when user click that link that equal with access [*http://0.0.0.0:3000/articles/1*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2F1%2Fedit&sa=D&sntz=1&usg=AFQjCNFuXIIzzToSXJHEADn-SyGT4xvrGA)*.*
2. Routes will find if that path has registered, if yes it will find controller *articles*and action show.
3. Then action show will find data base on parameter id that send to controller, after found it will appear in page view show.

Here we want to show data per-record this feature used when we want to see detail from list data. Now we want to modify index view, so the content of articles not showing all, open index.html.erb modify into:

<% @articles.each do |article| %>

<h3><%= article.title %></h3>

<p>

<%= article.content.truncate(20) %>

<%= link\_to 'read more', article\_path(article.id) %>

</p>

<h4>status : <%= article.status %></h4>

<hr/>

<% end %>

Here we add method helper *truncate* this function used for limit paragraph that will show in list. And add *link\_to*this function handle redirect page, so when click 'read more' link you will be redirected to show page. The question is where *article\_path*is come from, if you still remember about routes before, if we check with *rails routes:*

**Prefix          Verb          URI Pattern                   Controller#Action**

articles         GET          /articles(.:format)               articles#index

                   POST       /articles(.:format)               articles#create

new\_article   GET          /articles/new(.:format)        articles#new

edit\_article   GET          /articles/:id/edit(.:format)     articles#edit

article         GET          /articles/:id(.:format)           articles#show

                  PATCH     /articles/:id(.:format)           articles#update

                   PUT         /articles/:id(.:format)            articles#update

                   DELETE   /articles/:id(.:format)            articles#destroy

*article\_path* is come from prefix of action show. For example, if you want add link to new page get prefix *new\_article* then add *\_path,*so the path is **new\_article\_path**.

For complete resources can be found here:

[*https://guides.rubyonrails.org/v5.2/getting\_started.html#adding-links*](https://guides.rubyonrails.org/v5.2/getting_started.html#adding-links)

so when you open index page, you will see the read more link.

After we modify index view we want add query in controller, method show is a method that handle showing data in a view by specific id, open controller article and add:

def show

@article = *Article*.find\_by\_id(params[:id])

end

That will get data article base id that is send from the link, but we still not process *@article,* now open view show (*show.html.erb*), add this:

<h3><%= @article.title %></h3>

<p><%= @article.content %></p>

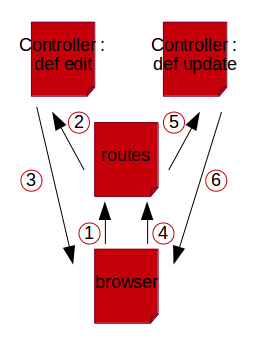
<p><%= link\_to 'back to index', articles\_path %></p>

Now try to click the read more link.

if you click link “back to index” it will redirect back to list page.

### Edit data

Work-flow Edit Data:



**Description:**

1. In the browser click edit link in list data, the data have id = 1, when user click that link that equal with access [*http://0.0.0.0:3000/articles/1/edit*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2F1%2Fedit&sa=D&sntz=1&usg=AFQjCNFuXIIzzToSXJHEADn-SyGT4xvrGA)*.*
2. Routes will check if that path is registered. If yes it will find controller article and action edit.
3. In controller will get parameter id=1, then used it to find data on table article. After success it will assign data to input form in view *edit.*
4. When click button submit it will send data to path [*http://0.0.0.0:3000/articles/1*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2F1&sa=D&sntz=1&usg=AFQjCNGx0imUj9AyJCehK18eqmVTWW2eSg), with method *put.*
5. Routes will check if that path registered, if found it will find controller article and action *update.*
6. In controller action *update* will process parameter that send from input form edit, if success it will redirect to index but if fails it will stay in page edit.

Process edit record is same with show data but different when process update, first we need modify method edit in controller. But before that we need add link list in view, open *index.html.erb* and modify into this:

<% @articles.each do |article| %>

<h3><%= article.title %></h3>

<p><%= article.content %></p>

<h4>status : <%= article.status %></h4>

<p>

<span><%= link\_to 'edit', edit\_article\_path(article.id) %><span>

</p>

<hr/>

<% end %>

After adding a link, we need setup method edit in controller, open article controller and add:

def edit

@article = *Article*.find\_by\_id(params[:id])

end

Then we need to setup input form in view of *edit.html.erb,*open the file and add:

<%= form\_for @article do |f| %>

<%= f.label :title %>

<%= f.text\_field :title %>

<br/>

<%= f.label :content %>

<%= f.text\_area :content, cols: 20, rows: 10 %>

<br/>

<%= f.label :status %>

<%= f.radio\_button :status, "active" %>

<%= label :article\_status, 'active' %>

<%= f.radio\_button :status, "off" %>

<%= label :article\_status, 'off' %>

<br/>

<%= f.submit :submit %>

<% end %>

<div>

<%= link\_to "back to index", articles\_path %>

</div>

the data is automatically populated in field input, but if you click submit it still not process edit, for process edit we need add update method, open controller article add:

def update

@article = *Article*.find\_by\_id(params[:id])

if @article.update(params\_article)

flash[:notice] = "Success Update Records"

redirect\_to action: 'index'

else

flash[:error] = "data not valid"

render 'edit'

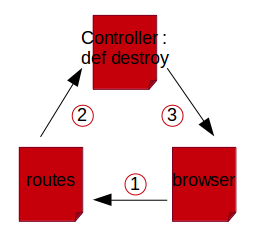
end

end

*params\_article*is private method that we have created before, this method is for handling process data from model article, so parameter article from form edit can be process.

### delete data

**Work-flow Delete Data:**



**Description:**

1. In browser click delete link in list data, the data have id = 1, when user click that link that equal with access [*http://0.0.0.0:3000/articles/1*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Farticles%2F1&sa=D&sntz=1&usg=AFQjCNGx0imUj9AyJCehK18eqmVTWW2eSg)*,*with method *DELETE.*
2. Routes will find if that path has registered, if yes it will find controller *articles*and action destroy.
3. In action destroy the parameter id used for searching data, then the data will be deleted. Then redirect to index page, either success or not.

For delete data we need link that send to path destory, open index view (*index.html.erb*), add link:

<% @articles.each do |article| %>

<h3><%= article.title %></h3>

<p><%= article.content %></p>

<h4>status : <%= article.status %></h4>

<p>

<span><%= link\_to 'edit', edit\_article\_path(article.id) %><span>

<span> | </span>

<span><%= link\_to 'delete', article\_path(article.id), method: "DELETE", data: {confirm: 'Delete this data ? '} %><span>

</p>

<hr/>

<% end %>

The different between show and delete is action GET and DELETE, if you check *rails route:*

**Prefix Verb URI Pattern Controller#Action**

articles GET /articles(.:format) articles#index

POST /articles(.:format) articles#create

new\_article GET /articles/new(.:format) articles#new

edit\_article GET /articles/:id/edit(.:format) articles#edit

article GET /articles/:id(.:format) articles#show

PATCH /articles/:id(.:format) articles#update

PUT /articles/:id(.:format) articles#update

**DELETE** /articles/:id(.:format) **articles#destroy**

- *method: “DELETE”,*is used for distinguish between link to show and delete, if you not specify the method for delete it will send you to show page.

- *data: {confirm: 'Delete this data ? '},*this is used for add confirmation before delete data.

For delete we didn't need a view like show, edit. What we need is method destroy in controller article, let open file articles\_controller.rb and add:

def destroy

@article = *Article*.find\_by\_id(params[:id])

if @article.destroy

flash[:notice] = "Success Delete a Records"

redirect\_to action: 'index'

else

flash[:error] = "fails delete a records"

redirect\_to action: 'index'

end

end

Now if you click delete link it will show popup dialog. if you click cancel button, process delete will break but if you click button ok delete process will continue.

## Adding Style, Custom Style and Serving Images (rails 5)

### Bootstrap 4

Put these gems into your gemfile:

# bootstrap

gem 'bootstrap', '~> 4.3.1'

#jquery

gem 'jquery-rails'

Now open your app/assets/javascripts/application.js, and add these lines:

//= require jquery3

//= require popper

//= require bootstrap-sprockets

Next, open your app/assets/stylesheets/application.scss (if the extension still css just rename it into scss), and add this line:

 @import "bootstrap";

Now if the server is running, restart your server.

### Custom stylesheet (same with js)

You can use the application.scss or add your own custom css file, for example you create custom.css just put that file into app/assets/stylesheets/ folder and open the config/initializers/assets.rb and add this line

# Precompile additional assets.

# application.js, application.css, and all non-JS/CSS in the app/assets

# folder are already added.

# Rails.application.config.assets.precompile += %w( admin.js admin.css )

Rails.application.config.assets.precompile += %w( custom.css )

If the server still running just restart the server.

### Images

For images just put your image into /assets/images/ folder and you can use it in your application like this:

<%= image\_tag("gf") %>

More info: <https://guides.rubyonrails.org/v5.2/asset_pipeline.html>

# Relationship

In ruby on rails relationship between table divide into three: one-to-one, one-to-many, & many-to-many. Before we talking about the three relationship let's create the new model and controller.

Generate model comment:

rails g model comment article\_id:integer user\_is:integer content:string status:string --skip-unit-test --no-test-framework

Don't forget run migration for create table comment, on terminal:

rails db:migrate

Generate controller comments:

rails g controller comments index new edit --no-helper --no-assets --no-controller-specs --no-view-specs --skip-unit-test --no-test-framework

Modify routes.rb, add restfull path for comment

open file routes.rb, change this code:

get 'comments/index'

get 'comments/new'

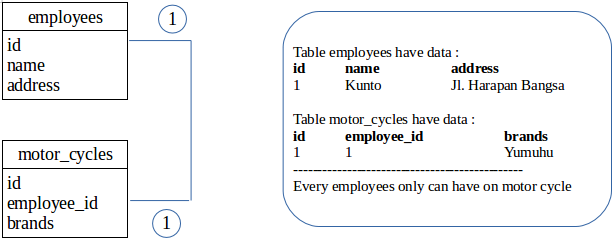
get 'comments/edit'

Into:

resources :comments

## One to One

One to one relationship is when a record in a table is exactly connect to one record in other table.



### Implement one-to-one in Ruby on Rails

**parent model:** has\_one <name\_relation\_singular>

**child model:** belongs\_to <name\_relation\_singular>

- model employee (*employee.rb*):

class Employee < *ActiveRecord::Base*

#name relation must singular

has\_one :motor\_cycle, dependent: :destroy

end

- model motor cycle (*motor\_cycle.rb*)

class MotorCycle < *ActiveRecord::Base*

#name relation must singular

belongs\_to :employee, dependent: :delete

end

### Using relation in query

**Action: insert**

**Active** **Record**:

<parent\_model\_name>.new(<column\_data>).<name\_relation>.build\_<name\_relation>(<column\_data).save

**Example**:

- Create new data:

@employee = *Employee*.new(name:"Warner", address: "Bandung").save

@employee.build\_motor\_cycle(brands: "Yumuhu Byson").save

**Action: select**

**Active** **Record**: <model\_name>.<option>.<name\_relation>

**Example**:

- Get data motorcycle from specific employee data:

*Employee*.find\_by\_id(1).motor\_cycle

- Get data employee from specific motorcycle data:

*MotorCycle*.find\_by\_id(1).employee

**Action: update data relation**

**Active** **Record**: <parent\_model\_name>.<option>.<name\_relation>.update(<data\_column>)

**Example**:

- Update relation data:

*Employee*.last.motor\_cycle.update(brands: "Yumuhu Byson")

**Action: delete data relation**

**Active** **Record**: <parent\_model\_name>.<option>.delete

**Example**:

- Delete parent data and his relation data:

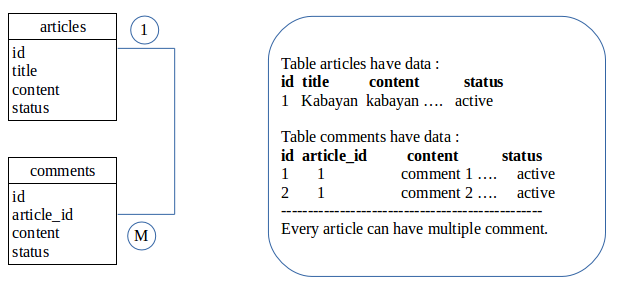
*Employee*.first.destroy

- Delete child data and his parent data:

*MotorCycle*.first.destroy

## One to Many

One to Many relationships is when a record in a table can be reference to many data by another table.



### Implement one-to-many in Ruby on Rails

**parent model:** has\_many <name\_relation\_plural>

**child model:** belongs\_to <name\_relation\_singular>

- Model article (*article.rb*) :

class Article < *ActiveRecord::Base*

#name relation must plural

has\_many :comments, dependent: :destroy

end

- Model comment (*comment.rb*)

class Comment < *ActiveRecord::Base*

#name relation must singular

belongs\_to :article

end

### Using relation in query

**Action: insert**

Active Record

**parent data:**

*<model\_name>.create(<column\_data>)*

**child data:**

*<parent\_model\_name>.<option>.<name\_relations>.create([{<column\_data>}, {<column\_data>}])*

Example:

- Create Parent data:

*Article*.create(title: "learn rails 4", content: "fell free write article", status: "not active")

- Create Child data:

*Article*.last.comments.create([{content: "comment 1"},{content: "comment 2"}])

**Action: select**

Active Record: *<model\_name>.<option>.<name\_relations>*

Example:

- Get all data comments from specific article data:

*Article*.find\_by\_id(1).comments

- Get data article from specific comment data:

*Comment*.find\_by\_id(1).article

**Action: update child**

**Active** **Record**:

*<parent\_model\_name>.<option>.<name\_relations>.<option>.update(<name\_column>)*

**Example**:

- Update child data:

*Article*.find\_by\_id(1).comments.first.update(content: "edited comment")

**action: delete relation**

**Active Record**: *<parent\_model\_name>.<option>.destroy*

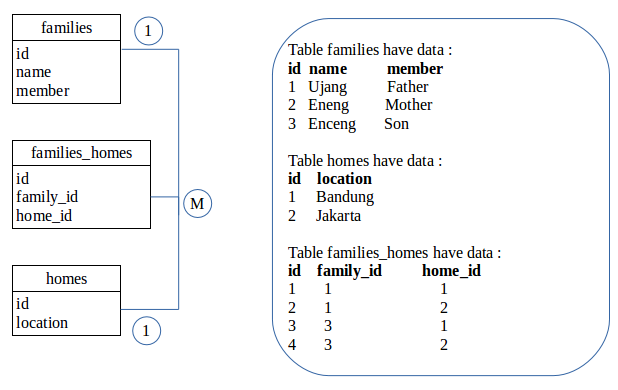
**Example**:

- Delete parent data and child data:

*Article*.first.destroy

## Many to Many

One to many is relationship between two table A and B in which A may contain a parent row for which there are many children in B and vice versa.



### Implement many-to-many in Ruby on Rails

**parent model:** has\_many <name\_relation\_plural>

**parent model:** has\_many <name\_relation\_plural>

**association model:** belongs\_to <name\_relation\_singular>

- model family (*family.rb*):

class Family < *ActiveRecord::Base*

#name relation must plural

has\_many :families\_homes

has\_many :homes, through: :families\_homes dependent: :destroy

end

- model home (*home.rb*)

class Home < *ActiveRecord::Base*

#name relation must plural

has\_many :families\_homes

has\_many :families, through: :families\_homes dependent: :destroy

end

- model family home (*families\_home.rb*)

class FamiliesHome < *ActiveRecord::Base*

#name relation must singular

belongs\_to :family

belongs\_to :home

end

### Using relation in query

**Action: insert**

Example:

- Create data:

@fam = *Family*.new(params[:family])

@fam.homes.create(params[:home])

**Action: update**

Example:

- Create data:

@fam = *Family*.find(params[:id])

@fam.update\_attributes(params[:family])

@fam.homes.build

**Action: select**

Example:

- Get all data homes from specific family data:

*Family*.find\_by\_id(1).homes

- Get data families from specific home data:

*Home*.find\_by\_id(1).families

For complete resources can be found here:

[*http://edgeguides.rubyonrails.org/association\_basics.html*](http://edgeguides.rubyonrails.org/association_basics.html)

# Authentication

## Create User

Before we start to create login form, we need to create user model.

Create model *user*:

rails g model user username:string email:string password:string --no-test-framework

Run migration to create table user, on terminal type:

rails db:migrate

Create controller *users*:

rails g controller users index new edit --no-helper --no-assets --no-test-framework

It will generate:

create app/controllers/users\_controller.rb

route get 'users/edit'

route get 'users/new'

route get 'users/index'

invoke erb

create app/views/users

create app/views/users/index.html.erb

create app/views/users/new.html.erb

create app/views/users/edit.html.erb

after successfully generating user’s controller, now we will modify route for user, open *routes.rb* then change:

get 'users/index'

get 'users/new'

get 'users/edit'

Into this:

get "sign\_up" => "users#new", :as => "sign\_up"

resources :users

When you check rails routes it will result:

**Prefix           Verb           URI Pattern              Controller#Action**

sign\_up        GET          /sign\_up(.:format)          users#new

users           GET          /users(.:format)             users#index

                   POST        /users(.:format)             users#create

new\_user     GET          /users/new(.:format)       users#new

edit\_user     GET          /users/:id/edit(.:format)    users#edit

user            GET          /users/:id(.:format)          users#show

                  PATCH      /users/:id(.:format)          users#update

                  PUT          /users/:id(.:format)          users#update

                  DELETE    /users/:id(.:format)          users#destroy

- sign\_up, is path for user new, so when you access *[http://0.0.0.0:3000/sign\_up](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Fsign_up&sa=D&sntz=1&usg=AFQjCNE3P6wsH78NVY4HTbWdQtROrCjRNQ)*it will show form for new user. This equal to [*http://0.0.0.0:3000/users/new*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Fusers%2Fnew&sa=D&sntz=1&usg=AFQjCNFTAyzwQBl38y1W1E2uuuUCajP6OA)*.*

Create new user data to table is need form input, process in controller and validation in model. First we will modify method new, open user controller file, lets open it, and change into:

def new

@user = *User*.new

end

Let’s open user new view, *new.html.erb*, modify existing code to:

<%= form\_for @user do |f| %>

<%= f.label :username %>

<%= f.text\_field :username %>

<%= @user.errors[:username].first unless @user.errors[:username].blank? %>

<br/>

<%= f.label :email %>

<%= f.text\_field :email %>

<%= @user.errors[:email].first unless @user.errors[:email].blank? %>

<br/>

<%= f.label :password %>

<%= f.password\_field :password %>

<%= @user.errors[:password].first unless @user.errors[:password].blank? %>

<br/>

<%= f.label :password\_confirmation %>

<%= f.password\_field :password\_confirmation %>

<%= @user.errors[:password\_confirmation].first unless @user.errors[:password\_confirmation].blank? %>

<br/>

<%= f.submit %>

<% end %>

After setup view and controller let create validation in model user, open *user.rb* add:

validates :username, presence: true, uniqueness: true

validates :email, presence: true, uniqueness: true

validates :password, presence: true, on: :create,

length: { minimum: 5 },

confirmation: true

Then try to access [*http://0.0.0.0:3000/sign\_up*](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Fsign_up&sa=D&sntz=1&usg=AFQjCNE3P6wsH78NVY4HTbWdQtROrCjRNQ)

If you try to submit valid data it will give you an error, this happens because action create in user controller still not setup. Let’s create that action, open user’s controller adds:

def create

@user = *User*.new(params\_user)

if @user.save

flash[:notice] = "Success Add Records"

redirect\_to root\_url

else

flash[:error] = "data not valid"

render "new"

end

end

private

def params\_user

params.require(:user).permit(:username, :email, :password, :password\_confirmation)

end

Don't forget in rails 4, parameter treated as strong\_parameter, that mean you must white list all parameter that will process by create action. Now try to insert the valid data again, if success it will save to user table, but another problem is we can see the password barely:



So it need encryption to hide the real password. We will use gem bcrypt-ruby, this gem base on gem bcrypt. With bcrypt-ruby we will generate password from combination salt (is random data that is used as an additional input to a one-way function that use by password) and password that inputted by user so it will result hash (one-way encryption so impossible for inverted) password.

First we need add bcrypt-ruby to gemfile, open *Gemfile* add:

gem 'bcrypt-ruby', require: 'bcrypt'

Then install that gem to our application, open terminal:

bundle install

Before implement bcrypt-ruby we need add column to table user for salt and modify column password, open terminal and run:

rails g migration add\_salt\_hash\_to\_users

That will create:

invoke active\_record

create db/migrate/20140912124348\_add\_salt\_hash\_to\_users.rb

open file migration and modify code to:

class AddSaltHashToUsers < *ActiveRecord::Migration*

def change

add\_column :users, :password\_salt, :string

rename\_column :users, :password, :password\_hash

end

end

Run migration:

rails db:migrate

After setup migration now we will implement bcrypt-ruby, open model user (user.rb), and add code:

class User < *ActiveRecord::Base*

attr\_accessor :password

before\_save :add\_salt\_and\_hash

validates :username, presence: true, uniqueness: true

validates :email, presence: true, uniqueness: true

validates :password, presence: true, on: :create,

length: { minimum: 5 },

confirmation: true

def add\_salt\_and\_hash

unless password.blank?

self.password\_salt = *BCrypt*::*Engine*.generate\_salt

self.password\_hash = *BCrypt*::*Engine*.hash\_secret(password, password\_salt)

end

end

end

- *attr\_accessor*, is property, for setter and getter.

- *before\_save*, is tell model user before save data run method *add\_salt\_and\_hash*

- *self.* , is for make sure to recognize password\_salt and password\_hash is part of model user.

Try add new data user from sign up form, if success you will get hash password and salt:



for more resources about bcrypt-ruby can be found here:

<https://github.com/codahale/bcrypt-ruby>

## Recaptcha

Recaptcha used for preventing bots so this will make sure a real human only that can register to the application. Here we will be using gem humanizer. Open Gemfile add:

gem 'humanizer'

Install gem:

bundle install

Implement humanizer in application, first generate humanizer, open terminal:

rails g humanizer

Then on model user (*user.rb*), inside class user:

include Humanizer

require\_human\_on :create

Next in new user view, inside block *form\_for* add:

<%= f.label :humanizer\_answer, @user.humanizer\_question %>

<%= f.text\_field :humanizer\_answer %>

<%= f.hidden\_field :humanizer\_question\_id %>

The last is setup strong parameter in controller user (users\_controller.rb), modify function params\_user to:

private

def params\_user

params.require(:user).permit(:username, :email, :password, :password\_confirmation, :humanizer\_answer, :humanizer\_question\_id)

end

Now try to access [http://0.0.0.0:3000/sign\_up](http://www.google.com/url?q=http%3A%2F%2F0.0.0.0%3A3000%2Fsign_up&sa=D&sntz=1&usg=AFQjCNE3P6wsH78NVY4HTbWdQtROrCjRNQ)

After password confirmation is input for answer captcha question.

for more resource about recaptcha:

<https://github.com/kiskolabs/humanizer>

\*after try several gem for recaptcha, gem humanizer that worked well for rails 4.1.4, you can try experiment with try other gem for recaptcha. Is better using image than text for recaptcha.

## Confirmation

Email confirmation used for making sure that email inserted for the user account is not fake, so user must activate the account from email that send when successfully sign up. In this topic we will use SMTP (simple mail transfer protocol).

First , we need setup smtp connection, this configuration put in (*config/environments/development.rb*), open that file add:

#config mailer with port 465

config.action\_mailer.raise\_delivery\_errors = true

config.action\_mailer.delivery\_method = :smtp

config.action\_mailer.smtp\_settings = {

:address => "smtp.gmail.com",

:port => 465,

:domain => "gmail.com",

:user\_name => "geeksfarmmailer",

:password => "geeks123farm",

:authentication => "login",

:ssl => true,

:tls => true,

:enable\_starttls\_auto => true

}

Or if you want connect without ssl use this configuration:

#config mailer with port 587

config.action\_mailer.raise\_delivery\_errors = true

config.action\_mailer.delivery\_method = :smtp

config.action\_mailer.smtp\_settings = {

:address => "smtp.gmail.com",

:port => 587,

:domain => 'gmail.com',

:user\_name => 'geeksfarmmailer',

:password => 'geeks123farm',

:authentication => 'plain',

:enable\_starttls\_auto => true

}

After setup configuration for email now we need setup class and view for email, rails provided generator to generate email configuration, on terminal type:

rails g mailer confirmation\_mailer

That will create:

create app/mailers/confirmation\_mailer.rb

invoke erb

create app/views/confirmation\_mailer

* *confirmation\_mailer.rb*, this class used for handling mailer method.
* *confirmation\_mailer*, this folder used for putting template view of email.

More resources is here: [***http://edgeguides.rubyonrails.org/action\_mailer\_basics.html***](http://edgeguides.rubyonrails.org/action_mailer_basics.html)

First create method for send email, open up *confirmation\_mailer.rb,* modify into this:

class ConfirmationMailer < *ActionMailer::Base*

def confirm\_email(*target\_email*, *activation\_token*)

@activation\_token = activation\_token

mail(to: target\_email,

         body: "http://localhost:3000/sessions/#{@activation\_token}/edit",

         content\_type: "text/html",

         subject: "Test Confirmation!",

         template\_path: "confirmation\_mailer",

         template\_name: "confirm\_email")

end

end

end

* *confirmation\_email* function will use when user account successfully created, now we need to setup view. Parameter target\_email will be getting from table user and activation\_token will be getting from generator in model user.
* *render 'confirm\_email'*, this view gets from *confirmation\_mailer* folder, let’s create *confirm\_email.html.erb* inside that folder, and add this html tag and rails link, *app/views/confirmation\_mailer/confirm\_email.html.erb*:

<h1>For activation your account Click link below : </h1>

<%= link\_to "Activate Your Account", "http://0.0.0.0:3000/sessions/#{@activation\_token}/edit", method: 'GET' %>

* *"http://0.0.0.0:3000/sessions/#{@activation\_token}/edit", method: 'GET',*is link that use for handle process confirmation email. Now we need setup route for session, open *routes.rb* add:

resources :sessions

* Then check if that routes are correct, on terminal type:

rails routes

* This the result:

**Prefix Verb URI Pattern Controller#Action**

sessions GET /sessions(.:format) sessions#index

POST /sessions(.:format) sessions#create

new\_session GET /sessions/new(.:format) sessions#new

edit\_session **GET** **/sessions/:id/edit(.:format)** sessions#edit

session GET /sessions/:id(.:format) sessions#show

PATCH /sessions/:id(.:format) sessions#update

PUT /sessions/:id(.:format) sessions#update

DELETE /sessions/:id(.:format) sessions#destroy

* After setup routes now we need create sessions controller, on terminal type:

rails g controller sessions --no-helper --no-assets --no-controller-specs --no-view-specs --skip-unit-test --no-test-framework

* Now we need add method edit in controller session, open *sessions\_controller.rb*, add:

def edit

user = *User*.find\_by\_activation\_token(params[:id])

if user.try(:update,{activation\_token: "", activation\_status: "active"})

flash[:notice] = "Your account has active"

redirect\_to root\_url

else

flash[:notice] = "Welcome to Rails 5"

redirect\_to root\_url

end

end

* *activation\_token* and *activation\_status*, is new columns that must add to table users, open terminal type:

rails generate migration add\_activation\_token\_status\_to\_users

* Open file migration modifies to this code:

class AddActivationTokenToUsers < *ActiveRecord::Migration*

def change

add\_column :users, :activation\_token, :string

add\_column :users, :activation\_status, :string

end

end

* Then execute the migration:

rails db:migrate

After adding column migration now we need to setup default value for activation\_status and token, this default value will add before new data user created, that will setup in **model user**, add this code inside **class user**:

before\_create :add\_activation\_token

def add\_activation\_token

self.activation\_token = *SecureRandom*.urlsafe\_base64

self.activation\_status = "not activated"

end

* SecureRandom, is an interface for secure random number generator.
* urlsave\_base64, is action for generate secure url number base on 64 string.

For complete resources: [***http://apidock.com/ruby/SecureRandom/urlsafe\_base64/class***](http://apidock.com/ruby/SecureRandom/urlsafe_base64/class)

That function will automatic running when new data user is added, it will generate token and status “not activated”. After we setup all configuration for email notification now we need implement mail class to action create in controller users, open *users\_controller.rb* modify action create to:

def create

@user = *User*.new(params\_user)

if @user.save

begin

*ConfirmationMailer*.confirm\_email("#{@user.email}", @user.activation\_token).deliver

rescue

flash[:notice] = "activation instruction fails send to your email"

end

flash[:notice] = "activation instruction has send to #{@user.email}"

redirect\_to root\_url

else

flash[:error] = "data not valid"

render "new"

end

end

Now try add new data user, if success it will create new user data then send activation to user email, first check the data user in database in the last column you will see token and status account.

## Login

After success creating activation email now the application need to implement the password and status, so user cannot access all web-page and only several page that allow for public. That can be implemented using session (temporary data that hold in browser).

From last topic we have create controller sessions that process activation account, now we will be using this controller too for login and logout, open *sessions\_controller.rb,*add:

def new

end

Then add view for login in sessions view, add *views/sessions/****new.html.erb,***and add this code:

<%= form\_tag sessions\_path do %>

<%= label\_tag :username %>

<%= text\_field\_tag :username, (params[:username] unless params[:username].blank?) %>

<br/>

<%= label\_tag :password %>

<%= password\_field\_tag :password %>

<br/>

<%= submit\_tag :submit %>

<% end %>

* *form\_tag*, is helper rails that allow us for process input form tag that don't rely on ActiveRecord. So, in controller we must manually process the params input.

For complete resources:

[***http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html***](http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html)

After creating form login, we need an action to be created for processing the form create session, open *sessions\_controller.rb* add:

def create

username = params[:username]

password = params[:password]

user = *User*.where("username =? and activation\_status =?", username, "active").first

user\_password = *BCrypt*::*Engine*.hash\_secret(password, user.password\_salt) unless user.blank?

if !user\_password.blank? and user.password\_hash.eql? user\_password

session[:user] = user.id

flash[:notice] = "Welcome #{user.username}"

redirect\_to root\_url

else

params[:username]

flash[:error] = "Your data not valid"

render "new"

end

end

* *session[:user] = user.id,*this is use for setting value of session with *user* name. This session will use when checking if user login or not.
* *user = User.where(....*, this query will check if username and status is active is exist, if found it will return array.
* *.first,*is function for catch first data from array.

Now we need a function for checking if session is present, open *applications\_controller.rb*and modify into:

class ApplicationController < *ActionController::Base*

# Prevent CSRF attacks by raising an exception.

# For APIs, you may want to use :null\_session instead.

protect\_from\_forgery with: :exception

helper\_method :current\_user, :check\_current\_user

def current\_user

current\_user ||= *User*.where("id = ? and activation\_status = ?", session[:user], 'active').first

end

def check\_current\_user

if current\_user.blank?

flash[:error] = "plase login first before run the action"

redirect\_to new\_session\_url

else

current\_user

end

end

end

* *helper\_method*, this used for declaring a controller method as a helper, so the method can access from all view, controller, model. Helper method useful when you need a method that can be access from everywhere inside the application.

For more resources:

[***http://apidock.com/rails/AbstractController/Helpers/ClassMethods/helper\_method***](http://apidock.com/rails/AbstractController/Helpers/ClassMethods/helper_method)

[***http://apidock.com/rails/v4.0.2/AbstractController/Helpers/ClassMethods/helper***](http://apidock.com/rails/v4.0.2/AbstractController/Helpers/ClassMethods/helper)

* *def current\_user*, this for populate value user base on current session.
* *def check\_current\_user*, this function used for checking if current user exists.

After create custom helper, now we need implement that function in articles controller, so only user with session that can access page new, edit and action delete. Open *article\_controller.rb*, inside class ArticlesController add:

before\_action :check\_current\_user, only: [:new, :create, :edit, :update, :destroy]

* *before\_action*, this will run before execute all action in controller. o*nly:,*used for action new, create, edit, update, destroy. So when you access or click link for edit, delete it need login first before do that action.

For more resources:

[***http://stackoverflow.com/questions/16519828/rails-4-before-filter-vs-before-action***](http://stackoverflow.com/questions/16519828/rails-4-before-filter-vs-before-action)

now we need to add link for login and logout in the root page, open *views/articles/index.html.erb*add this code:

<h4>

<% if !current\_user %>

<%= link\_to 'Log-in', new\_session\_path %>

<% else %>

Hello <%= current\_user.username %>

<%= link\_to 'Log-Out', session\_path(current\_user.id), method: "DELETE" %>

<% end %>

</h4>

The last step is to add function for logout in controller sessions, open *sessions\_controller.rb* add:

def destroy

session[:user] = nil

flash[:notice] = "logout session success"

redirect\_to root\_url

end

Now try to login.

TROUBLESHOOT

If using smtp with Gmail raise this error: ***"Net::SMTPAuthenticationError: 534-5.7.14 ..."***, that means we need to update Gmail security setting and change it to low, go to this page:  
- <http://www.google.com/accounts/DisplayUnlockCaptcha>

- <https://www.google.com/settings/security/lesssecureapps>

# Ajax

for ajax implementation we will used show article page, so when user insert a comment user will directly see list of comment that currently inserted without refreshing the page, but if save fails the error message will be shown in the form input area (refresh only area form input).

First let modify method show in *articles\_controller.rb*, add this:

def show

@article = *Article*.find\_by\_id(params[:id])

@comments = @article.comments.order("id desc")

@comment = *Comment*.new

end

* *@comments,* it will use by list data of comment in show view.
* @comment, it will create new object for create new comments.

Now let’s setup view show article (views/articles/show.html.erb), add this:

<h3><%= @article.title %></h3>

<p><%= @article.content %></p>

<p><%= link\_to 'back to index', articles\_path %></p>

<div id="form-comment">

<%= render 'form' %>

</div>

<h4>Here the comments</h4>

<div id="list-comments">

<%= render 'comments' %>

</div>

* *render 'form'*, is where partial view for input form comment is loaded (*\_form.html.erb*).
* render 'comments, is where partial view for list data comment is show (\_comments.html.erb).

Create new file to: views/*comments/\_form.html.erb,* and add this:

<%= form\_for([@article, @article.comments.build], remote:true, html: {id: :form\_id })  do |f| %>

  <div class="form-group">

    <%= f.label :commenter %><br>

    <%= f.text\_field :commenter, id: "text\_commenter", class: "form-control" %>

  </div>

  <div class="form-group">

    <%= f.label :body %><br>

    <%= f.text\_area :body, id: "text\_body", rows: "5", class: "form-control" %>

  </div>

  <%= f.submit class: "btn btn-outline btn-outline-info"  %>

<% end %>

* *@comment*, is global variable that declared from action show in controller articles, use for creating new object.
* *remote: true*, where you tell to controller comments the request type is js/ajax, so when you click button submit it won reload/refresh the current page.

Create new file to: views/*comments/\_comment.html.erb,* and add this:

<p>

  <strong>Commenter:</strong>

  <%= comment.commenter %>

</p>

<p>

  <strong>Comment:</strong>

  <%= comment.body %>

</p>

<p>

  <%= link\_to 'Destroy Comment', [comment.article, comment],

               method: :delete,

               data: { confirm: 'Are you sure?' }, class: "btn btn-outline btn-outline-warning" %>

</p>

* *@comment*, is global variable that declared from action show in controller articles, use for hold all data comment for specific article.

After setup the view now we need to setup default value for status and add validation to content of comment. Open model *comment.rb,* and add:

class Comment < *ActiveRecord::Base*

belongs\_to :article

   validates :body, presence: true, length: { minimum: 10 }

end

* *validation :body, presence: true,*is validation for input comment.

Let’s add process to controller comments, open *comments\_controller.rb*and modify into:

class CommentsController < ApplicationController

  before\_action :check\_current\_user, only: [:new, :create, :edit, :update, :destroy]

  def create

    respond\_to do |format|

      # render plain: @article.inspect

      @article = Article.find(params[:article\_id])

      @comment = @article.comments.create(comment\_params)

      # render plain: @comment.inspect

      if @comment.save

        format.js { @article }

      else

        format.js { @article = Article.find(params[:article\_id]) }

      end

    end

    # redirect\_to article\_path(@article)

  end

  def destroy

    @article = Article.find(params[:article\_id])

    @comment = @article.comments.find(params[:id])

    @comment.destroy

    redirect\_to article\_path(@article)

  end

  private

  def comment\_params

    params.require(:comment).permit(:commenter, :body)

  end

end

* *check\_current\_user*, is used for authentication.
* *respond\_to*, is action for define type of response, in this case we use *format.js* that will render the page as js. So after success or fails saving data it will render page *create.js.erb*.
* *comment\_params*, is strong variable so you need manually add specific column that will insert to table.

Create new file to: views/comments*/create.js.erb,* and add this:

<% if @comment.errors.present? %>

    $("#form-comment").html(" <%= j( render 'comments/form' ) %>");

    $($(".field\_with\_errors")[0]).append("<p class = 'message'> <%= escape\_javascript( @comment.errors[:content].first ) %></p>");

<% else %>

    $("#list-comments").html(" <%= j( render @article.comments.order("id desc") ) %>");

    $("#text\_commenter").val('');

    $("#text\_body").val('');

    $("div.field\_with\_errors p.message").remove();

<% end %>

* *@comment.errors.present*, will check if error happens when create new comments.
* *render 'comments/form'* and *render @article.comments.order(“id desc”)'*, that will render the partial page from views/comments.

Now try the feature that we’ve created.

# Filtering

## Pagination

Pagination in Ruby on Rails can be achieved in various way but the easy is using gems service like will\_paginate, kaminari. In this example will using gem 'kaminari' for handle pagination.

### Implementation

* Install gem 'kaminari', open file Gemfile from your Rails project, and add gem 'kaminari':

gem 'kaminari'

save that file, then run 'bundle install' in console for install kaminari.

* Setup controller, open articles\_controller.rb and modify method index become:

def index

@articles = *Article*.page(params[:page]).per(10)

end

Description:

-> .page(params[:page]), use for handling request page number from request that send by link in view.

-> .per(10), use for limiting number of data that will send to view from controller.

* Setup view, open page views/articles/index.html.erb, then add cod for showing number page after looping articles declaration:

...

<% @articles.each do |article| %>

......

<% end %>

<%= paginate @articles %>

</tbody>

To generate kaminari styles, we can use:

rails generate kaminari:views bootstrap4

**Available themes**: bootstrap2, bootstrap3, bootstrap4, bourbon, bulma, foundation, foundation5, github, google, materialize, purecss, semantic\_ui.

Complete resource:

- <https://github.com/amatsuda/kaminari>

## Searching

Searching useful for getting data base on specific keyword. In this example we will be using JQuery AJAX for filtering data.

### Implementation

For implementation we will using menu articles at page index.

* Setup view, modify code at page views/articles/index.html.erb, into below code:

<div class="row">

  <div class="col-md-6"><h1>Articles Lists</h1></div>

  <div class="col-md-6"><%= link\_to 'New article', new\_article\_path, class: "btn btn-outline btn-outline-info float-right" %></div>

</div>

<div class="row mt-3">

  <form id="get\_search">

    <div class="form-group">

      <input class="form-control-sm" type="search" placeholder="Type here to search" aria-label="Search" id="search" name="search">

      <input class="btn btn-sm btn-outline-info" type='button' onclick="submitFrmAjax();" value="Search">

    </div>

  </form>

</div>

<div class="row" id="article\_list">

  <%= render @articles %>

  <%= paginate @articles %>

</div>

<%= javascript\_include\_tag 'custom' %>

Description:

-> <input type="search" ...., use for input keyword search.

-> id="article\_list", will used by ajax as anchor for populate data articles from result search query.

-> <%= render @articles %>, for render partial page article that will automatic loop code.

-> <%= javascript\_include\_tag 'custom' %>, will use for ajax code.

* For render record using <%= render @articles %>, we need partial page with name \_article.html.erb. Create file partial in views/articles/\_article.html.erb, and add this code:

<div class="post-preview">

  <a href="<%= article\_path(article) %>">

    <h2 class="post-title">

      <%= article.title %>

    </h2>

    <h3 class="post-subtitle">

      <%= article.text %>

    </h3>

  </a>

  <p class="post-meta">Posted by

    <a href="#">User</a>

  </p>

</div>

* Setup AJAX, open file assets/javascripts/custom.js (don’t forget to register it into config/initializers/assets.rb)

# Rails.application.config.assets.precompile += %w( admin.js admin.css )

Rails.application.config.assets.precompile += %w( custom.css custom.js )

* if not exist let’s create that file first, and add:

function submitFrmAjax() {

  Rails.ajax({

    url: "/articles",

    type: "GET",

    data: $("#get\_search").serialize()

  });

}

- Setup controller, open articles\_controller.rb and modify query in method index into:

def index

    @articles = Article.page(params[:page]).per(2)

    respond\_to do |format|

      if params[:search].present?

        format.js {

          @articles = Article.where("title like ? or text like ?", "%#{params[:search]}%", "%#{params[:search]}%").page(params[:page]).per(2)

        }

        format.html

      else

        format.html

      end

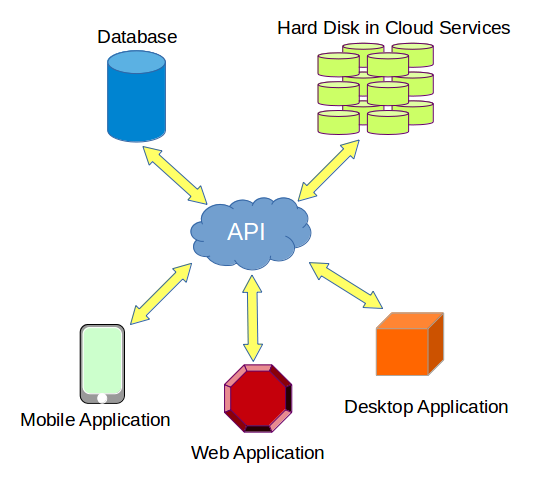
    end

  end

* Setup response from format.js, create file /views/articles/index.js.erb, and add this code:
* $('#article\_list').html("<%= j(render @articles) %>");

# Rest API (Application Programming Interface)

API is a bridge that connect between resources (like database, hard disk) and application, which allow us for sharing of resources with many applications.   
  
Example of API Work-flow:



## Description

From above example API control two resources (Database, Cloud Service), and API will be called as “Provider”. The applications (Mobile, Web, Desktop) will called as “Consumers”, every Consumers that will access resources from Provider will be checked for their privileges as consumer.

**Success Case:**

1. Consumers will send request with a token as an authentication.
2. Provider will check if the token is valid, if yes the provider will send response 200.
3. After authentication success Provider will check if Consumers have the authorization to resources, if yes Provider will check if resources that requested is present, if yes send response 200, with JSON containing requested resource, but if not found provider will send response 404.

**Fails Case:**

1. Consumers will send request with a token as an authentication.
2. Provider will check if the token is valid, if not Provider will send response 401.

A good api must have these 4 traits:

1. **predictable and consistent**: code reuse for api end points.
2. **static**: an api should not change in a breaking way.
3. **simple and clear**: an api should return what’s expected.
4. **flexible**: easy to scale and maintain

## RESTful

RESTful (Representational State Transfer) is an architecture for API design, that communicate over HTTP request and response. API request in modern style usually use AJAX and the result value usually using JSON format.

Method of RESTful API:

|  |  |  |  |
| --- | --- | --- | --- |
| METHOD | ACTION | DESCRIPTION | URI |
| GET | Select | This method will select all resources that need by Consumers. | http://domain.ok/resources |
| PUT | Update | This method will update resources that targeted by Consumers. | http://domain.ok/resources/item-no |
| POST | Create | This method will save new resources that send by Consumers. | http://domain.ok/resources |
| DELETE | Delete | This method will remove all resources that target by Consumers. | http://domain.ok/resource/item-no |

*Complete resources:*

* <http://en.wikipedia.org/wiki/Application_programming_interface>
* <http://en.wikipedia.org/wiki/Web_API>
* <http://en.wikipedia.org/wiki/Representational_state_transfer>
* <http://en.wikipedia.org/wiki/List_of_HTTP_status_codes>

## Ruby on Rails API

Best Practice

Study case:

I have a project that need to manipulate the "User" data via an API, and I need JWT auth before I can use some of the endpoint’s.

The endpoints are:

|  |  |  |  |
| --- | --- | --- | --- |
| URL / ENDPOINT | METHOD | AUTH | DESCRIPTION |
| /auth/login | POST | False | Generate token |
| /auth/signup | POST | False | Register new user |
| /users | GET | False | Return all users |
| /users/{username} | GET | True | Return user |
| /users/{username} | PUT | True | Update user |
| /users/{username} | DELETE | True | Destroy user |

Steps…

1. Generate rails project for api. Type this on your terminal:

rails new rails-jwt --api

1. Setup the database and execute:

rails db:create

1. Add Json Web Token (JWT) and bcrypt gem in **gemfile:**

# Use ActiveModel has\_secure\_password

gem 'bcrypt', '~> 3.1.7'

# Use Json Web Token (JWT) for token-based authentication

gem 'jwt'

1. Install dependencies by typing this on your terminal:

bundle install

1. Update routes:

Rails.application.routes.draw do

  resources :users, param: :\_username

  post "/auth/login", to: "authentication#login"

  post "/auth/signup", to: "authentication#signup"

  get "/\*a", to: "application#not\_found"

  # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html

end

In the routes.rb, we defined routes for users using resources. resources syntax helps us for generating REST API pattern path for user using \_username as parameter. So, it will look like our REST API table above.

1. Create **json\_web\_token.rb** class in **app/lib/**

class JsonWebToken

  SECRET\_KEY = Rails.application.secrets.secret\_key\_base.to\_s

  def self.encode(payload, exp = 24.hours.from\_now)

    payload[:exp] = exp.to\_i

    JWT.encode(payload, SECRET\_KEY)

  end

  def self.decode(token)

    decoded = JWT.decode(token, SECRET\_KEY)[0]

    HashWithIndifferentAccess.new decoded

  end

end

SECRET\_KEY is the key for encoding and decoding token. In the code above, we assign secret key that generated by default by rails application into SECRET\_KEY variable. SECRET\_KEY must be secret and not to be shared. Everytime we’re doing some encoding and decoding using JWT, we need to specify the SECRET\_KEY. By grouping and encapsulating JWT encoding and decoding mechanism in this class, we will reduce couple of code that have responsibility for doing encoding and decoding job, because we don’t need to specify SECRET\_KEY everytime. Decode and encode function above defined as static function because it will give a flexibility for doing encoding and decoding job without instantiate JsonWebToken object.

self.encode function has 2 parameters. first payload and second exp. payload is key-value object for holding data that want to be encoded. exp stand for expiration for holding expiration time token. if exp not specified it will give you default value in 24 hour or one day.

In self.decode function we decoded the token that given by user and get the first value then assign to decoded variable, the first value contain payload that we had already encoded before and second value contain information about algorithm that we use for encoding and decoding token.

1. Create **authorize\_request function** in **app/controllers/ApplicationController.rb**

class ApplicationController < ActionController::API

  def not\_found

    render json: { error: "not\_found" }

  end

  def authorize\_request

    header = request.headers["Authorization"]

    header = header.split(" ").last if header

    begin

      @decoded = JsonWebToken.decode(header)

      @current\_user = User.find(@decoded[:user\_id])

    rescue ActiveRecord::RecordNotFound => e

      render json: { errors: e.message }, status: :unauthorized

    rescue JWT::DecodeError => e

      render json: { errors: e.message }, status: :unauthorized

    end

  end

end

authorize\_request function has responsibility for authorizing user request. first, we need to get token in header with ‘Authorization’ as key. with this token now we can decode and get the payload value. in this application we define user\_id in payload. You should not include the user credentials data into payload because it will cause security issue, you can include data that needed to authorizing user. When performing JsonWebToken.decode function, it will return JWT::DecodeError if there was an error like token was expired, token not valid, token missing etc. After we got user\_id from payload then we will try to find user by id and assign it into current\_user variable, If user not exist it will return ActiveRecord::RecordNotFound and it will render error message with http status unauthorized.

1. Create **user model**:

rails g model user name:string username:string email:string password\_digest:string

1. Execute migration:

rails db:migrate

1. Add user validation in **user** **model**:

class User < ApplicationRecord

  has\_secure\_password

  #   mount\_uploader :avatar, AvatarUploader

  validates :email, presence: true, uniqueness: true

  validates :email, format: { with: URI::MailTo::EMAIL\_REGEXP }

  validates :username, presence: true, uniqueness: true

  validates :password,

            length: { minimum: 6 },

            if: -> { new\_record? || !password.nil? }

end

1. Create user controller:

rails g controller users

1. Add Create, Read, Update, Delete (CRUD) functionality to the **UsersController**:

class UsersController < ApplicationController

  before\_action :authorize\_request, except: :index

  before\_action :find\_user, except: %i[create index]

  # GET /users

  def index

    @users = User.all

    render json: @users, status: :ok

  end

  # GET /users/{username}

  def show

    render json: @user, status: :ok

  end

  # POST /users

  def create

    @user = User.new(user\_params)

    if @user.save

      render json: @user, status: :created

    else

      render json: { errors: @user.errors.full\_messages },

             status: :unprocessable\_entity

    end

  end

  # PUT /users/{username}

  def update

    unless @user.update(user\_params)

      render json: { errors: @user.errors.full\_messages },

             status: :unprocessable\_entity

    end

  end

  # DELETE /users/{username}

  def destroy

    @user.destroy

  end

  private

  def find\_user

    @user = User.find\_by\_username!(params[:\_username])

  rescue ActiveRecord::RecordNotFound

    render json: { errors: "User not found" }, status: :not\_found

  end

  def user\_params

    params.permit(

      :avatar, :name, :username, :email, :password, :password\_confirmation

    )

  end

end

1. Create authentication controller:

rails g controller authentication

1. Implement login and signup features:

class AuthenticationController < ApplicationController

  before\_action :authorize\_request, except: [:login, :signup]

  # POST /auth/login

  def login

    @user = User.find\_by\_email(params[:email])

    if @user&.authenticate(params[:password])

      token = JsonWebToken.encode(user\_id: @user.id)

      time = Time.now + 24.hours.to\_i

      render json: { token: token, exp: time.strftime("%m-%d-%Y %H:%M"),

                    username: @user.username }, status: :ok

    else

      render json: { error: "unauthorized" }, status: :unauthorized

    end

  end

  # POST /auth/signup

  def signup

    @user = User.find\_by\_email(params[:email])

    if @user

      render json: { error: "email already taken" }, status: :unauthorized

    else

      @user = User.new(user\_params)

      @user.save

      render json: { user: @user }, status: :created

    end

  end

  private

  def login\_params

    params.permit(:email, :password)

  end

  def user\_params

    params.permit(

      :name, :username, :email, :password

    )

  end

end

In JWT there is no way to invalidate token, you can use one of this approach to implement logout feature:

1. Remove token from client, but token still valid, in my opinion you should use short time period token.
2. Add token into blacklist, when token added into blacklist token still valid until expiration time but you can deny this request from accessing resource.

Now you can test your API endpoints with API Tester Client (ex: curl, talend, postman, etc.), in this manner signup, login, get user by name, get all user.

## Active Model Serializer

Active model serializer it’s a fabulous gem, that’s brings convention over configuration to our API’s JSON, he works with serializers and adapters and I’ll show you how to use.

First of all, open your gemfile and add:

# use model serializer to format return data

gem 'active\_model\_serializers'

And run bundle update

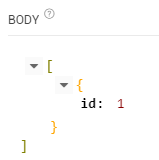
After this, you can create any serializer with the following command:

rails g serializer user

Rails output:

create  app/serializers/user\_serializer.rb

Now if we restart our server and make the same GET http://localhost:3000/users will we see this output:



Don’t panic, this is normal! We’ll see only Ids of the users because we don’t set up our user serializer to output others attributes, to fix this, open your app/serializers/user\_serializer.rb and you will see only this:

class UserSerializer < ActiveModel::Serializer

  attributes :id

end

The only attribute that is outputing is id and we need to add more things, add the others things of the model that you want to show, and my code will look like:

class UserSerializer < ActiveModel::Serializer

  attributes :id, :name, :username, :email, :created\_at

end

Now if we call our API again, we will get:



### Custom Methods on Serializer

One thing is troubling me on this response, it’s created\_at format, look on the last json, is not good for me, let’s fix this, on the UserSerializer add this method:

class UserSerializer < ActiveModel::Serializer

  attributes :id, :name, :username, :email, :created\_at

  def created\_at

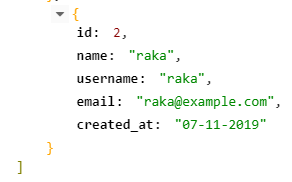
    object.created\_at.strftime "%d-%m-%Y"

  end

end

With this, we’re taking the object that is book record in this context and formatting date, remember that you can add any custom method using this, just remember to add in the attributes on the top of your serializer and make your custom method.

Now our return json is looking like this:



### Json API Adapter

One of the advantages of use active model serializer gem is that this gem already has an adapter to use jsonapi.org convention in your API, for who don’t know, these shared conventions pattern the json responses. I really recommend that you read more about here.

Create a file named **active\_record\_serializer.rb** at **config/initializers/** and put this inside:

ActiveModelSerializers.config.adapter = :json\_api

Restart your application, make the same call for your API and see the difference.

|  |  |
| --- | --- |
| Before | After |
|  |  |

Your api response are now following jsonapi specification without pains.

## CORS

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.

An example of a cross-origin request: the front-end JavaScript code served from https://domain-a.com uses XMLHttpRequest to make a request for https://domain-b.com/data.json.

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. For example, XMLHttpRequest and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request resources from the same origin the application was loaded from, unless the response from other origins includes the right CORS headers.

To handle this, we must enable cors in our API.

1. Use rack-cors gem add this in gemfile gem 'rack-cors'
2. Execute bundle install
3. Add configuration to config/application.rb

*module* YourApp

*class* Application < Rails::Application

*# ...*

*# Rails 5*

    config.middleware.insert\_before 0, Rack::Cors *do*

      allow *do*

        origins "\*"

        resource "\*", headers: :any, methods: [:get, :post, :options]

*end*

*end*

*# Rails 3/4*

    config.middleware.insert\_before 0, "Rack::Cors" *do*

      allow *do*

        origins "\*"

        resource "\*", headers: :any, methods: [:get, :post, :options]

*end*

*end*

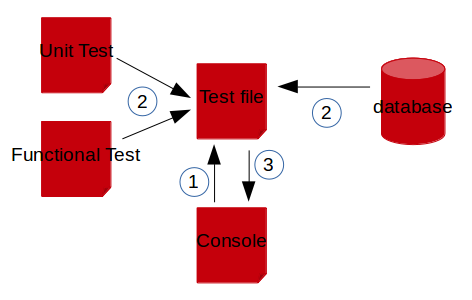
*end*

*end*

We use insert\_before to make sure Rack::Cors runs at the beginning of the stack to make sure it isn't interfered with by other middleware (see Rack::Cache note in Common Gotchas section).

# Testing

## Work-flow



**Description:**

1. In console user run command for executing a test file.
2. - Test categories divide into two type:

      Unit test, this test will be testing all functions in model class.

      Functional test, this test will be testing all action in controller class like:

      -- Whether login is success or not.

      -- Whether action responses are correct.

      -- Whether the process of select, save, edit, create, & delete is working.

      -- Whether the return message of an action is right.

    - If test need sample data it will call data from '<name\_project>\_test', that is database for testing, and the data that use in test will not change (create, edit, delete).

1. After a successful run, it will report the test result.

**Study Case**

Here we will be using unit test and functional test, and for model and controller that will be test is the “articles” that has been created in the last session, and for details:

1. create dummy data in database with, environment test, and seed data for test environment.
2. create test file inside folder '<name\_application>/test'.

First we will see the file model and controller:

model articles, '*article.rb*’:

class Article < *ActiveRecord::Base*

has\_many :comments, dependent: :destroy

validates :title, presence: true, length: { minimum: 5 }

validates :content, presence: true, length: { minimum: 10 }

validates :status, presence: true

scope :status\_active, -> {where(status: 'active')}

end

From above model we will create **Unit test**, detail:

* check if relation with comment is working.
* validation for *column title, content, status*and make sure create and edit data will fail if data is blank.
* validate *title,* *content* and make sure create and edit will fail if data is not reach minimum character.
* check if custom scope is working.

Controller articles, 'articles\_controller':

class ArticlesController < *ApplicationController*

before\_action :check\_current\_user, only: [:new, :create, :edit, :update, :destroy]

def index

@articles = *Article*.status\_active

end

def new

@article = *Article*.new

end

def create

@article = *Article*.new(params\_article)

if @article.save

flash[:notice] = "Success Add Records"

redirect\_to action: 'index'

else

flash[:error] = "data not valid"

render 'new'

end

end

def show

@article = *Article*.find\_by\_id(params[:id])

@comments = @article.comments.order("id desc")

@comment = *Comment*.new

end

def edit

@article = *Article*.find\_by\_id(params[:id])

end

def update

@article = *Article*.find\_by\_id(params[:id])

if @article.update(params\_article)

flash[:notice] = "Success Update Records"

redirect\_to action: 'index'

else

flash[:error] = "data not valid"

render 'edit'

end

end

def destroy

@article = *Article*.find\_by\_id(params[:id])

if @article.destroy

flash[:notice] = "Success Delete a Records"

redirect\_to action: 'index'

else

flash[:error] = "fails delete a records"

redirect\_to action: 'index'

end

end

private

def params\_article

params.require(:article).permit(:title, :content, :status)

end

end

From above controller we will create **Functional test**, detail:

* Create method for create session, this is used by every test method in this test because in controller require login before execute action create, edit, delete.
* Check if action index working: response success/200, will render index view page.
* Check if action new working: response success/200, will render new view page.
* Check if action create working: response with redirect/300, redirect to root view page, send success message.
* Check if action show working: response success/200, will render show view page.
* Check if action edit working: response success/200, render edit view page.
* Check if action update working: response with redirect/300, redirect to root view page, send success message.

## `Implementation

Make sure you have move inside rails app, lets create database test, on terminal run:

rails db:create test

rails db:migrate RAILS\_ENV=test

After create database and table now we need create seed data, on terminal run:

mkdir db/seeds/

touch db/seeds/test.rb

Above code will create folder *seeds*and file *test.rb.* Now open file test.rb and add this:

puts "========= Seeding Articles Data ========="

*ActiveRecord*::*Base*.connection.execute("TRUNCATE articles")

*Article*.create!([

{

id: 1,

title: "Seed Title One",

content: "Content of article seed one",

status: "active"

},

{

id: 2,

title: "Seed Title Two",

content: "Content of article seed two",

status: "active"

}

])

puts "========= Seeding Comments Data ========="

*ActiveRecord*::*Base*.connection.execute("TRUNCATE comments")

*Comment*.create!([

{

id: 1,

article\_id: 1,

content: "First comment for article seed one",

status: "active"

},

{

id: 2,

article\_id: 1,

content: "Second comment for article seed one",

status: "active"

},

{

id: 3,

article\_id: 2,

content: "First comment for article seed two",

status: "active"

}

])

puts "========= Seeding Users Data ========="

*ActiveRecord*::*Base*.connection.execute("TRUNCATE users")

*User*.create!([

{

id: 1,

email: "warner@gmail.com",

username: "warner",

password: "12345"

},

{

id: 2,

email: "develror@gmail.com",

username: "develror",

password: "12345"

}

])

*User*.update\_all(activation\_token: "", activation\_status: "active")

Save it, and we need setup environment seed, open file '*db/seeds.rb*', and add:

load(*Rails*.root.join( 'db', 'seeds', "#{*Rails*.env.downcase}.rb"))

Save it, and run in console:

rails db:seed RAILS\_ENV=test

## Unit Test

After successfully prepare data for database test, now we will create Unit test, on terminal run:

touch test/models/article\_test.rb

Then open that file and add:

require 'test\_helper'

class ArticleTest < *ActiveSupport::TestCase*

#--------------Start Test Relation--------------

test "Get all comment for first article" do

article = *Article*.first

assert article.comments

end

test "Get article from first comment" do

comment = *Comment*.first

assert comment.article

end

#---------------Start Test validation----------------

# \*\*\*\*\*\*\*\*\*\*when save data\*\*\*\*\*\*\*\*\*\*

test "create article with invalid, title is blank" do

article = *Article*.new(title: "", content: "lorem ipsum heroda kara",status: "active")

assert\_equal false, article.valid?

assert\_not article.save

end

test "create article with invalid data, title < 5" do

article = *Article*.new(title: "abcd", content: "lorem ipsum heroda kara",status: "active")

assert\_equal false, article.valid?

assert\_not article.save

end

test "create article with invalid data, content is blank" do

article = *Article*.new(title: "learn uni test", content: "",status: "active")

assert\_equal false, article.valid?

assert\_not article.save

end

test "create article with invalid data, content < 10" do

article = *Article*.new(title: "learn uni test", content: "abcde fg",status: "active")

assert\_equal false, article.valid?

assert\_not article.save

end

test "create article with invalid data, status is blank" do

article = *Article*.new(title: "learn unit test", content: "conten of article learn unit test",status: "")

assert\_equal false, article.valid?

assert\_not article.save

end

test "create article with valid data" do

article = *Article*.new(:title => "Testing", :content => "This is body", status: 'active')

assert\_equal true, article.valid?

assert article.save

end

# \*\*\*\*\*\*\*\*\*\*when edit data\*\*\*\*\*\*\*\*\*\*\*\*

test "edit article with invalid, title is blank" do

article = *Article*.first

assert\_equal false, article.update(title: "", content: "lorem ipsum heroda kara",status: "active")

end

test "edit article with invalid data, title < 5" do

article = *Article*.first

assert\_equal false, article.update(title: "abcd", content: "lorem ipsum heroda kara",status: "active")

end

test "edit article with invalid data, content is blank" do

article = *Article*.first

assert\_equal false, article.update(title: "learn uni test", content: "",status: "active")

end

test "edit article with invalid data, content < 10" do

article = *Article*.first

assert\_equal false, article.update(title: "learn uni test", content: "abcde fg",status: "active")

end

test "edit article with invalid data, status is blank" do

article = *Article*.first

assert\_equal false, article.update(title: "learn unit test", content: "conten of article learn unit test",status: "")

end

test "edit article with valid data" do

article = *Article*.first

assert\_equal true, article.update(title: "Edit Testing", content: "Edit This is body")

end

#----------------End Test Validation-----------------

#--------------Test custom Scope-----------------

test "check scope, data exist if status active" do

assert *Article*.status\_active

end

end

* require 'test\_helper', is helper that use by test file, for create your own helper test it can add in '*test/test\_helper.rb*'.
* *class ArticleTest < ActiveSupport::TestCase*, that is how you declared the unit test class.
* *test '<description\_test>' do*

*<content\_test>*

*end*

that is a format for creating method in test file.

* *assert*, is method for test that make sure if return is true.
* *assert\_equal*, is method for make sure condition is same as expected.

For complete resource about assertion method:

[***http://guides.rubyonrails.org/testing.html#available-assertions***](http://guides.rubyonrails.org/testing.html#available-assertions)

Execute Unit test, on terminal run:

rails test:units

Or

rails test:models

Or

rails test test/models/article\_test.rb

Or

ruby -I test test/models/article\_test.rb

a

That will result:

    Run options: --seed 56860

    # Running:

    ...............

    Finished in 3.388544s, 4.4267 runs/s, 6.1974 assertions/s.

    15 runs, 21 assertions, 0 failures, 0 errors, 0 skips

if you get above result that means your test success, without failures and errors.

## Functional Test

Let’s create file for functional test, on terminal run:

touch test/controllers/articles\_controller\_test.rb

Then open that file and add:

require 'test\_helper'

class ArticlesControllerTest < *ActionController::TestCase*

# def setup will call before every test

# this function for create session

def setup

create\_session "develror@gmail.com"

end

test "index should render correct template" do

get :index

assert\_response :success

assert\_template :index

assert\_template layout: "layouts/application"

end

test "should response 200 and render to new page" do

get :new

assert\_response :success

assert\_template :new

assert\_template layout: "layouts/application"

end

test "should create article and send flash success " do

post :create, article: {title: 'test controller title', content: 'test controller content', status: 'active'}

assert\_response :redirect

assert\_redirected\_to root\_path

assert\_equal "Success Add Records", flash[:notice]

end

test "should response 200 and render page show" do

get :show, id: *Article*.first.id

assert\_response :success

assert\_template :show

end

test "should response 200 and render to edit page" do

get :edit, id: *Article*.first.id

assert\_response :success

assert\_template :edit

end

test "should update article and send flash success " do

patch :update, id: *Article*.first.id, article: {title: 'edit test controller title', content: 'edit test controller content'}

assert\_response :redirect

assert\_redirected\_to root\_path

assert\_equal "Success Update Records", flash[:notice]

end

private

def create\_session(*email*)

user = *User*.find\_by\_email(email)

session[:user] = user.id unless user.nil?

end

end

* *def setup*, is method that will automatic run at first time every test method running.
* *assert\_response,* is method for check type response that return by test.
* *assert\_template*, is method for check if result is render specific page.
* *assert\_redirected\_to*, is method for check if result is redirect to specific page.
* *assert\_equal,* is method for make sure if condition test equal as your expected, in this example use as check if save success it will show flash notice.

Now execute the fuctional test:

ruby -I test test/controllers/articles\_controller\_test.rb

That will result:

    Run options: --seed 48504

    # Running:

    ......

    Finished in 6.192611s, 0.9689 runs/s, 3.2297 assertions/s.

    6 runs, 20 assertions, 0 failures, 0 errors, 0 skips

if you get above result that means your test is success.

For complete resources : [***http://guides.rubyonrails.org/testing.html***](http://guides.rubyonrails.org/testing.html)

# Deployment

## Install Passenger

Don’t forget to install Ruby before doing these. The following commands will install Passenger + Nginx through Phusion's APT repository. If you already had Nginx installed, then these commands will upgrade Nginx to Phusion's version (with Passenger compiled in).

# Install our PGP key and add HTTPS support for APT

sudo apt-get install -y dirmngr gnupg

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 561F9B9CAC40B2F7

sudo apt-get install -y apt-transport-https ca-certificates

# Add our APT repository

sudo sh -c 'echo deb https://oss-binaries.phusionpassenger.com/apt/passenger xenial main > /etc/apt/sources.list.d/passenger.list'

sudo apt-get update

# Install Passenger + Nginx

sudo apt-get install -y nginx-extras passenger

## Enable the Passenger Nginx module and restart Nginx.

Edit /etc/nginx/nginx.conf and uncomment include /etc/nginx/passenger.conf;. For example, you may see this:

include /etc/nginx/passenger.conf;

If you don't see a commented version of include /etc/nginx/passenger.conf; inside nginx.conf, then you need to insert it yourself. Insert it into /etc/nginx/nginx.conf inside the http block. For example:

...

http {

include /etc/nginx/passenger.conf;

...

}

When you are finished with this step, restart Nginx:

sudo service nginx restart

### Check installation

After installation, please validate the install by running sudo /usr/bin/passenger-config validate-install. For example:

sudo /usr/bin/passenger-config validate-install

\* Checking whether this Phusion Passenger install is in PATH... ✓

\* Checking whether there are no other Phusion Passenger installations... ✓

All checks should pass. If any of the checks do not pass, please follow the suggestions on screen.

Finally, check whether Nginx has started the Passenger core processes. Run sudo /usr/sbin/passenger-memory-stats. You should see Nginx processes as well as Passenger processes. For example:

sudo /usr/sbin/passenger-memory-stats

Version: 5.0.8

Date : 2015-05-28 08:46:20 +0200

...

---------- Nginx processes ----------

PID PPID VMSize Private Name

-------------------------------------

12443 4814 60.8 MB 0.2 MB nginx: master process /usr/sbin/nginx

12538 12443 64.9 MB 5.0 MB nginx: worker process

### Processes: 3

### Total private dirty RSS: 5.56 MB

----- Passenger processes ------

PID VMSize Private Name

--------------------------------

12517 83.2 MB 0.6 MB PassengerAgent watchdog

12520 266.0 MB 3.4 MB PassengerAgent server

12531 149.5 MB 1.4 MB PassengerAgent logger

...

If you do not see any Nginx processes or Passenger processes, then you probably have some kind of installation problem or configuration problem.

## Push your code to a Git repository

We want to transfer our application's code to the server. The easiest way to do that is via Git.

If you have already setup a Git repository, push your application's code to that repository by running this on your local computer:

### Login to your server, create a user for the app

Login to your server with SSH. Replace adminuser with the name of an account with administrator privileges or sudo privileges.

Now that you have logged in, you should create an operating system user account for your app. For security reasons, it is a good idea to run each app under its own user account, in order to limit the damage that security vulnerabilities in the app can do. Passenger will automatically run your app under this user account as part of its user account sandboxing feature.

You should give the user account the same name as your app. But for demonstration purposes, this tutorial names the user account myappuser.

sudo adduser myappuser

We also ensure that that user has your SSH key installed:

sudo mkdir -p ~myappuser/.ssh

touch $HOME/.ssh/authorized\_keys

sudo sh -c "cat $HOME/.ssh/authorized\_keys >> ~myappuser/.ssh/authorized\_keys"

sudo chown -R myappuser: ~myappuser/.ssh

sudo chmod 700 ~myappuser/.ssh

sudo sh -c "chmod 600 ~myappuser/.ssh/\*"

Install Git on the server and pull your code. You need to pick a location in which to permanently store your application's code. A good location is /var/www/APP\_NAME. Let us create that directory.

sudo mkdir -p /*var*/www/myapp

sudo chown myappuser: /*var*/www/myapp

Replace myapp and myappuser with your app's name and your app user account's name.

Now let us pull the code from Git:

cd /*var*/www/myapp

sudo -u myappuser -H git clone git://github.com/username/myapp.git code

Your app's code now lives on the server at /var/www/myapp/code.

## Preparing the app's environment

### Login as the app's user

All subsequent instructions must be run under the application's user account. While logged into your server, login under the application's user account as follows:

sudo -u myappuser -H bash -l

### Install app dependencies

Your application has various dependencies. They must be installed. Most of these dependencies are gems in your Gemfile, managed by Bundler. You can install them by running bundle install --deployment --without development test -j 2 in your app's directory:

cd /*var*/www/myapp/code

bundle install --deployment --without development test

Your app may also depend on services, such as PostgreSQL, Redis, etc. Installing services that your app depends on is outside of this tutorial's scope.

### Configure database.yml and secrets.yml

Since your Rails app probably needs a database, you need to edit config/database.yml. For demonstration purposes, we will setup your app with an SQLite database because that is the easiest.

Open the file:

nano config/database.yml

Ensure that the production section looks like this:

production:

adapter: sqlite3

database: db/production.sqlite3

Rails also needs a unique secret key with which to encrypt its sessions. Starting from Rails 4, this secret key is stored in config/secrets.yml. But first, we need to generate a secret key. Run:

bundle exec rails secret

This command will output a secret key. Copy that value to your clipboard. Next, open config/secrets.yml:

nano config/secrets.yml

If the file already exists, look for this:

production:

secret\_key\_base: <%=ENV["SECRET\_KEY\_BASE"]%>

Then replace it with the following. If the file didn't already exist, simply insert the following.

production:

secret\_key\_base: the value that you copied from 'rails secret'

To prevent other users on the system from reading sensitive information belonging to your app, let's tighten the security on the configuration directory and the database directory:

chmod 700 config db

chmod 600 config/database.yml config/secrets.yml

Compile Rails assets and run database migrations

Run the following command to compile assets for the Rails asset pipeline, and to run database migrations:

bundle exec rails assets:precompile db:migrate RAILS\_ENV=production

## Configuring Nginx and Passenger

Now that you are done with transferring your app's code to the server and setting up an environment for your app, it is time to configure Nginx so that Passenger knows how to serve your app.

### Determine the Ruby command that Passenger should use

We need to tell Passenger Which Ruby command it should use to run your app, just in case there are multiple Ruby interpreters on your system. Please run passenger-config about ruby-command to find out which Ruby interpreter you are using. For example:

passenger-config about ruby-command

passenger-config was invoked through the following Ruby interpreter:

Command: /usr/local/rvm/gems/ruby-2.3.3/wrappers/ruby

Version: ruby 2.3.3p85 (2015-02-26 revision 49769) [x86\_64-linux]

...

Please take note of the path after "Command" (in this example, /usr/local/rvm/gems/ruby-2.3.3/wrappers/ruby). You will need it in one of the next steps.

### Go back to the admin account

You have previously logged into your app's user account in order to prepare the app's environment. That user does not have sudo access. In the next steps, you need to edit configuration files, for which sudo access is needed. So you need to switch back to the admin account.

This can be done by simply exiting the shell that was logged into the app's user account. You will then be dropped back to the admin account. For example:

# This is what you previously ran:

admin$ sudo -u myappuser -H bash -l

myappuser$ ...

# Type `exit` to go back to the account you were before

myappuser$ exit

admin$ \_

### Edit Nginx configuration file

We need to create an Nginx configuration file and setup a virtual host entry that points to your app. This virtual host entry tells Nginx (and Passenger) where your app is located.

sudo nano /etc/nginx/sites-enabled/myapp.conf

Replace myapp with your app's name.

Put this inside the file:

server {

listen 80;

server\_name yourserver.com;

# Tell Nginx and Passenger where your app's 'public' directory is

root /*var*/www/myapp/code/public;

# Turn on Passenger

passenger\_enabled on;

passenger\_ruby /path-to-ruby;

}

Replace yourserver.com with your server's host name, and replace /var/www/myapp/code with your application's code directory path. However, make sure that Nginx is configured to point to the public subdirectory inside it!

Replace /path-to-ruby with the Ruby command that you obtained in step 3.1.

When you are done, restart Nginx:

sudo service nginx restart

### Test drive

You should now be able to access your app through the server's host name! Try running this from your local computer. Replace yourserver.com with your server's hostname, exactly as it appears in the Nginx config file's server\_name directive.

curl http://yourserver.com/

...your app's front page HTML...

If you do not see your app's front page HTML, then these are the most likely causes:

You did not correctly configure your server\_name directive. The server\_name must exactly match the host name in the URL. For example, if you use the command curl http://45.55.91.235/ to access your app, then the server\_name must be 45.55.91.235.

You did not setup DNS records. Setting up DNS is outside the scope of this tutorial. In the meantime, we recommend that you use your server's IP address as the server name.

## Transferring latest code

Login to your server with SSH:

Replace myappuser with name of the application's OS user account.

Starting from this point, unless stated otherwise, all commands that we instruct you to run should be run on the server, not on your local computer!

### Pull latest code from Git

Go to your application's code directory on the server, then use Git to pull the latest code:

cd /*var*/www/myapp/code

git pull

### Install app dependencies

Your application's gem dependencies may have changed, so we should install any updated gem dependencies. Run:

bundle install --deployment --without development test

### Compile Rails assets and run database migrations

If your app is a Rails app, then you need to compile the latest Rails assets and run any database migrations. If your app is not a Rails app, please skip to the next step.

bundle exec rails assets:precompile db:migrate RAILS\_ENV=production

### Restart application

Passenger may still be serving an old instance of your application. Now that all application updates have been prepared, tell Passenger to restart your application so that the updates take effect.

passenger-config restart-app $(pwd)

more info: <https://www.phusionpassenger.com/library/>