Ruby on Rails

- Server-side web application framework written in Ruby
- Model-view-controller (MVC) framework
- Providing default structures for a database, we pages and services
- Encourages and facilitates the use of web standards such:
    a. JSON or XML for data transfer
    b. HTML, CSS and JavaScript for display and user interfacing
- Emphasizes the use of other well-known software engineering patterns and paradigms

History:

- David Heinemeier Hansson removed Ruby on Rails from his work on the project management tool Basecamp at the web application

JULY 2004 and FEBRUARY 2005

- He releases Rails as an open source but did not share commit rights to the project until the year of 2005 in February.

AUGUST 2006 and OCTOBER 2007

- Ruby on Rails reached a milestone when Apple announced that it would ship this framework with Mac OS X v10.5 "Leopard", which was in October 2007.

MARCH 15, 2009

- They've released a new version of Rails which is Rails version 2.3 with major new developments in templates, engines, Rack and nested model forms.

DECEMBER 23, 2008

- Merb is another web application framework and announced it would work with the Merb project to bring the best ideas of it into Rails 3 ending the unnecessary duplication across both communities and this framework was merged with Rails.

Rails 3.1: Released on August 31, 2011, featuring Reversible Database Migrations, Asset Pipeline, Streaming, jQuery as default JavaScript library and newly introduced CoffeeScript and Sass into the stack.

Rails 3.2: Released on January 20, 2012 with a faster development mode and routing engine (also known as Journey engine), Automatic Query Explain and Tagged Logging. Rails 3.2.x is the last version that supports Ruby 1.8.7. Rails 3.2.12 supports Ruby 2.0.

Rails 4.0: Released on June 25, 2013, introducing Russian Doll Caching, Turbolinks, Live Streaming as well as making Active Resource, Active Record Observer and other components optional by splitting them as gems.

Rails 4.1: Released on April 8, 2014, introducing Spring, Variants, Enums, Mailer previews, and secrets.yml.

Rails 4.2: Released on December 19, 2014, introducing Active Job, asynchronous emails, Adequate Record, Web Console, and foreign keys.

Rails 5.0: Released on June 30, 2016, introducing Action Cable, API mode, and Turbolinks 5.

Rails 5.0.0.1: Released on August 10, 2016, with Exclusive use of rails CLI over Rake and supports Ruby 2.2.2+ versions

Rails 5.1: Released on April 27, 2017, introducing JavaScript integration changes (management of JavaScript dependencies from NPM via Yarn, optional compilation of JavaScript using Webpack, and a rewrite of Rails UJS to use vanilla JavaScript instead of depending on jQuery), system tests using Capybara, encrypted secrets, parameterized mailers, direct & resolved routes, and a unified form_with helper replacing the form_tag/form_for helpers.

Why Ruby?
Ruby originated in Japan and now it is gaining popularity in US and Europe as well. The following factors contribute towards its popularity:

- Easy to learn
- Open source (very liberal license)
- Rich libraries
- Very easy to extend
- Truly object-oriented
- Less coding with fewer bugs
- Helpful community

Although we have many reasons to use Ruby, there are a few drawbacks as well that you may have to consider before implementing Ruby:

1. Performance Issues – Although it rivals Perl and Python, it is still an interpreted language and we cannot compare it with high-level programming languages like C or C++.

2. Threading model – Ruby does not use native threads. Ruby threads are simulated in the VM rather than running as native OS threads.

Rails Strengths
Rails is packed with features that make you more productive, with many of the following features building on one other.

1. Metaprogramming

Where other frameworks use extensive code generation from scratch, Rail framework uses Metaprogramming techniques to write programs. Ruby is one of the best languages for Metaprogramming, and Rails uses this capability well. Rails also uses code generation but relies much more on Metaprogramming for the heavy lifting.

2. Active Record
Rails introduces the Active Record framework, which saves objects into the database. The Rails version of the Active Record discovers the columns in a database schema and automatically attaches them to your domain objects using metaprogramming.

3. Convention over configuration
Most web development frameworks for .NET or Java force you to write pages of configuration code. If you follow the suggested naming conventions, Rails doesn't need much configuration.

4. Scaffolding
You often create temporary code in the early stages of development to help get an application up quickly and see how major components work together. Rails automatically creates much of the scaffolding you'll need.

6. Built-in testing
Rails creates simple automated tests you can then extend. Rails also provides supporting code called harnesses and fixtures that make test cases easier to write and run. Ruby can then execute all your automated tests with the rake utility.

7. Three environments
Rails gives you three default environments: development, testing, and production. Each behaves slightly differently, making your entire software development cycle easier. For example, Rails creates a fresh copy of the Test database for each test run.

Frameworks:

A framework is a program, set of programs, and/or code library that writes most of your application for you. When you use a framework, your job is to write the parts of the application that make it do the specific things you want.

When you set out to write a Rails application, leaving aside the configuration and other housekeeping chores, you have to perform three primary tasks –

- Describe and model your application's domain – The domain is the universe of your application. The domain may be a music store, a university, a dating service, an address book, or a hardware inventory. So here you have to figure out what's in it, what entities exist in this universe and how the items in it relate to each other. This is equivalent to modeling a database structure to keep the entities and their relationship.

- Specify what can happen in this domain – The domain model is static; you have to make it dynamic. Addresses can be added to an address book. Musical scores can be purchased from music stores. Users can log in to a dating service. Students can register for classes at a university. You need to identify all the possible scenarios or actions that the elements of your domain can participate in.

- Choose and design the publicly available views of the domain – At this point, you can start thinking in Web-browser terms. Once you've decided that your domain has students, and that they can register for classes, you can envision a welcome page, a registration page, and a confirmation page, etc. Each of these pages, or views, shows the user how things stand at a certain point.

Based on the above three tasks, Ruby on Rails deals with a Model/View/Controller (MVC) framework.

Ruby on Rails MVC Framework
The Model View Controller principle divides the work of an application into three separate but closely cooperative subsystems.

Model (ActiveRecord)
It maintains the relationship between the objects and the database and handles validation, association, transactions, and more.

This subsystem is implemented in ActiveRecord library, which provides an interface and binding between the tables in a relational database and the Ruby program code that manipulates database records. Ruby method names are automatically generated from the field names of database tables.

View (ActionView)
It is a presentation of data in a particular format, triggered by a controller's decision to present the data. They are script-based template systems like JSP, ASP, PHP, and very easy to integrate with AJAX technology.

This subsystem is implemented in ActionView library, which is an Embedded Ruby (ERb) based system for defining presentation templates for data presentation. Every Web connection to a Rails application results in the displaying of a view.

Controller (ActionController)
The facility within the application that directs traffic, on the one hand, querying the models for specific data, and on the other hand, organizing that data (searching, sorting, messaging it) into a form that fits the needs of a given view.

This subsystem is implemented in ActionController, which is a data broker sitting between ActiveRecord (the database interface) and ActionView (the presentation engine).

Pictorial Representation of MVC Framework
Given below is a pictorial representation of Ruby on Rails Framework –

Rails Framework
Directory Representation of MVC Framework
Assuming a standard, default installation over Linux, you can find them like this –

```
tp> cd /usr/local/lib/ruby/gems/2.2.0/gems
tp> ls
```

You will see subdirectories including (but not limited to) the following –

```
actionpack-x.y.z
activerecord-x.y.z
rails-x.y.z
```

Over a windows installation, you can find them like this –

```
tp>cd ruby\lib\ruby\gems\2.2.0\gems
ruby\lib\ruby\gems\2.2.0\gems\>dir
```

You will see subdirectories including (but not limited to) the following –

MVC
ActionView and ActionController are bundled together under ActionPack.

ActiveRecord provides a range of programming techniques and shortcuts for manipulating data from an SQL database. ActionController and ActionView provides facilities for manipulating and displaying that data. Rails ties it all together.

Migrations
Rails Migration allows you to use Ruby to define changes to your database schema, making it possible to use a version control system to keep things synchronized with the actual code.

This has many uses, including –

- Teams of developers – If one person makes a schema change, the other developers just need to update, and run "rake migrate".
- Production servers – Run "rake migrate" when you roll out a new release to bring the database up to date as well.

- Multiple machines – If you develop on both a desktop and a laptop, or in more than one location, migrations can help you keep them all synchronized.

What Can Rails Migration Do?

- create_table(name, options)
- drop_table(name)
- rename_table(old_name, new_name)
- add_column(table_name, column_name, type, options)
- rename_column(table_name, column_name, new_column_name)
- change_column(table_name, column_name, type, options)

- remove_column(table_name, column_name)
- add_index(table_name, column_name, index_type)
- remove_index(table_name, column_name)

Migrations support all the basic data types – The following is the list of data types that migration supports –

- string – for small data types such as a title.
- text – for longer pieces of textual data, such as the description.
- integer – for whole numbers.
- float – for decimals.
- datetime and timestamp – store the date and time into a column.
- date and time – store either the date only or time only.
- binary – for storing data such as images, audio, or movies.
- Boolean – for storing true or false values.

Valid column options are – The following is the list of valid column options.

- limit (:limit => "50")
- default (:default => "blah")
- null (:null => false implies NOT NULL)

NOTE – The activities done by Rails Migration can be done using any front-end GUI or directly on SQL prompt, but Rails Migration makes all those activities very easy.

See the Rails API for details on these.

Create the Migrations

Here is the generic syntax for creating a migration –

```
application_dir> rails generate migration table_name
```

This will create the file db/migrate/001_table_name.rb. A migration file contains the basic Ruby syntax that describes the data structure of a database table.

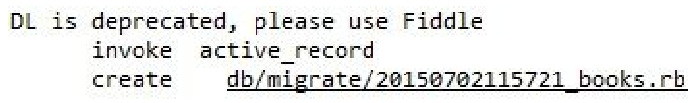
NOTE – Before running the migration generator, it is recommended to clean the existing migrations generated by model generators.

We will create two migrations corresponding to our three tables – *books and subjects*.

Books migration should be as follows –

```
tp> cd library
library> rails generate migration books
```

Above command generates the following code.

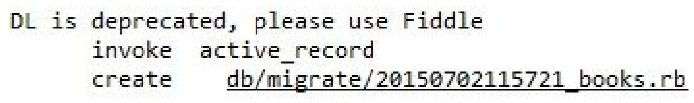


```
DL is deprecated, please use Fiddle
      invoke  active_record
      create    db/migrate/20150702115721_books.rb
```

subject migration should be as follows –

```
tp> cd library
library> rails generate migration subjects
```

Above command generates the following code.



```
DL is deprecated, please use Fiddle
      invoke  active_record
      create    db/migrate/20150702115721_books.rb
```

Notice that you are using lower case for book and subject and plural form while creating migrations. This is a Rails paradigm that you should follow each time you create a Migration.

Edit the Code

Go to db/migrate subdirectory of your application and edit each file one by one using any simple text editor.

Modify 001_books.rb as follows –

The ID column will be created automatically, so don't do it here as well.

```
class Books < ActiveRecord::Migration

  def self.up
    create_table :books do |t|
      t.column :title, :string, :limit => 32, :null => false
      t.column :price, :float
      t.column :subject_id, :integer
      t.column :description, :text
      t.column :created_at, :timestamp
    end
  end

  def self.down
    drop_table :books
  end
end
```

The method self.up is used when migrating to a new version, self.down is used to roll back any changes if needed. At this moment, the above script will be used to create *books* table.

Modify 002_subjects.rb as follows –

```
class Subjects < ActiveRecord::Migration
  def self.up

    create_table :subjects do |t|
      t.column :name, :string
    end

    Subject.create :name => "Physics"
    Subject.create :name => "Mathematics"
    Subject.create :name => "Chemistry"
    Subject.create :name => "Psychology"
    Subject.create :name => "Geography"
  end

  def self.down
    drop_table :subjects
  end
end
```

The above script will be used to create *subjects* table and will create five records in the subjects table.

Run the Migration

Now that you have created all the required migration files. It is time to execute them against the database. To do this, go to a command prompt and go to the library directory in which the application is located, and then type rake migrate as follows –

```
library> rake db:migrate
```

This will create a "schema_info" table if it doesn't exist, which tracks the current version of the database - each new migration will be a new version, and any new migrations will be run until your database is at the current version.

Rake is a Ruby build program similar to Unix *make* program that Rails takes advantage of, to simplify the execution of complex tasks such as updating a database's structure etc.

Running Migrations for Production and Test Databases

If you would like to specify what Rails environment to use for the migration, use the RAILS_ENV shell variable.

For example –

```
library> export RAILS_ENV = production
library> rake db:migrate
library> export RAILS_ENV = test
library> rake db:migrate
library> export RAILS_ENV = development
library> rake db:migrate
```

NOTE – In Windows, use "set RAILS_ENV = production" instead of *export*command.

Ajax stands for Asynchronous JavaScript and XML. Ajax is not a single technology; it is a suite of several technologies. Ajax incorporates the following –

- XHTML for the markup of web pages
- CSS for the styling

- Dynamic display and interaction using the DOM
- Data manipulation and interchange using XML
- Data retrieval using XMLHttpRequest
- JavaScript as the glue that meshes all this together

Ajax enables you to retrieve data for a web page without having to refresh the contents of the entire page. In the basic web architecture, the user clicks a link or submits a form. The form is submitted to the server, which then sends back a response. The response is then displayed for the user on a new page.

When you interact with an Ajax-powered web page, it loads an Ajax engine in the background. The engine is written in JavaScript and its responsibility is to both communicate with the web server and display the results to the user. When you submit data using an Ajax-powered form, the server returns an HTML fragment that contains the server's response and displays only the data that is new or changed as opposed to refreshing the entire page.

For a complete detail on AJAX you can go through our AJAX Tutorial

How Rails Implements Ajax

Rails has a simple, consistent model for how it implements Ajax operations. Once the browser has rendered and displayed the initial web page, different user actions cause it to display a new web page (like any traditional web application) or trigger an Ajax operation −

- Some trigger fires − This trigger could be the user clicking on a button or link, the user making changes to the data on a form or in a field, or just a periodic trigger (based on a timer).

- The web client calls the server – A JavaScript method, *XMLHttpRequest*, sends data associated with the trigger to an action handler on the server. The data might be the ID of a checkbox, the text in an entry field, or a whole form.
- The server does processing – The server-side action handler (Rails controller action)-- does something with the data and returns an HTML fragment to the web client.
- The client receives the response – The client-side JavaScript, which Rails creates automatically, receives the HTML fragment and uses it to update a specified part of the current page's HTML, often the content of a <div> tag.

These steps are the simplest way to use Ajax in a Rails application, but with a little extra work, you can have the server return any kind of data in response to an Ajax request, and you can create custom JavaScript in the browser to perform more involved interactions.

AJAX Example

This example works based on scaffold, Destroy concept works based on ajax.

In this example, we will provide, list, show and create operations on ponies table. If you did not understand the scaffold technology then we would suggest you to go through the previous chapters first and then continue with AJAX on Rails.

Creating An Application

Let us start with the creation of an application It will be done as follows –

```
rails new ponies
```

The above command creates an application, now we need to call the app directory using with cd command. It will enter in to an application directory then we need to call a scaffold command. It will be done as follows –

```
rails generate scaffold Pony name:string profession:string
```

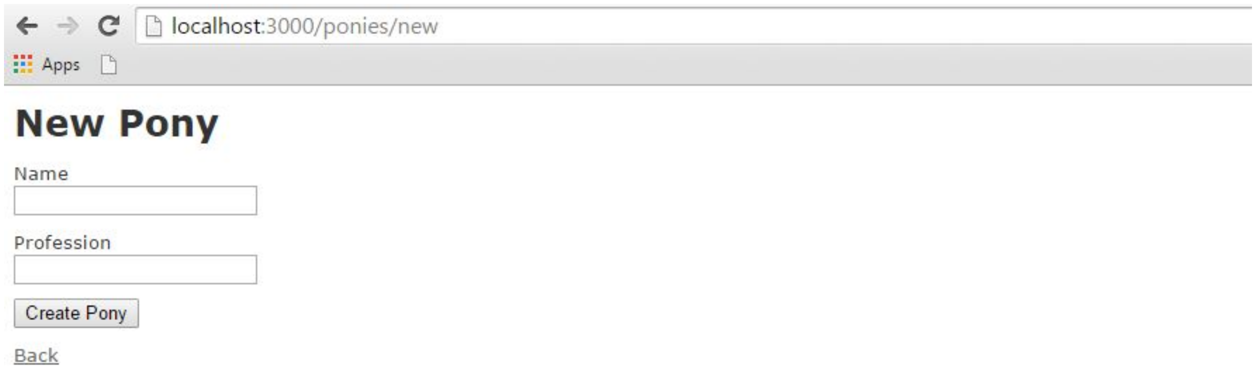
Above command generates the scaffold with name and profession column. We need to migrate the data base as follows command

```
rake db:migrate
```

Now Run the Rails application as follows command

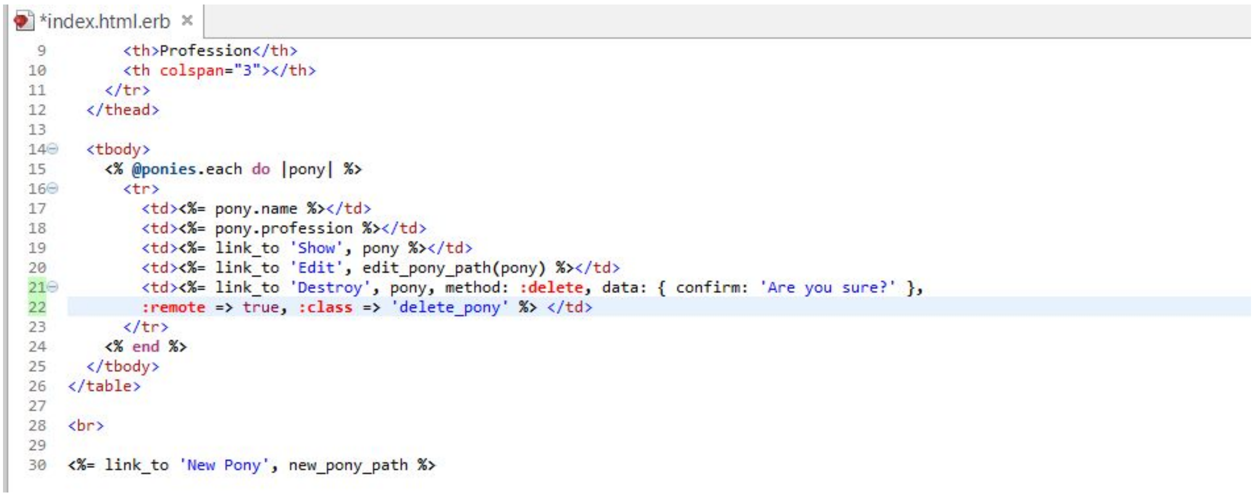
```
rails s
```

Now open the web browser and call a url as http://localhost:3000/ponies/new, The output will be as follows



Creating an Ajax

Now open app/views/ponies/index.html.erb with suitable text editors. Update your destroy line with :remote => true, :class => 'delete_pony'.At finally, it looks like as follows.



*index.html.erb

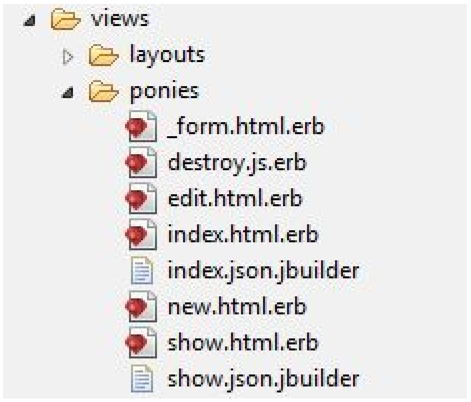
```
      <th>Profession</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @ponies.each do |pony| %>
      <tr>
        <td><%= pony.name %></td>
        <td><%= pony.profession %></td>
        <td><%= link_to 'Show', pony %></td>
        <td><%= link_to 'Edit', edit_pony_path(pony) %></td>
        <td><%= link_to 'Destroy', pony, method: :delete, data: { confirm: 'Are you sure?' },
        :remote => true, :class => 'delete_pony' %> </td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New Pony', new_pony_path %>
```

Create a file, destroy.js.erb, put it next to your other .erb files (under app/views/ponies). It should look like this –



```
views
  layouts
  ponies
    _form.html.erb
    destroy.js.erb
    edit.html.erb
    index.html.erb
    index.json.jbuilder
    new.html.erb
    show.html.erb
    show.json.jbuilder
```

Now enter the code as shown below in destroy.js.erb

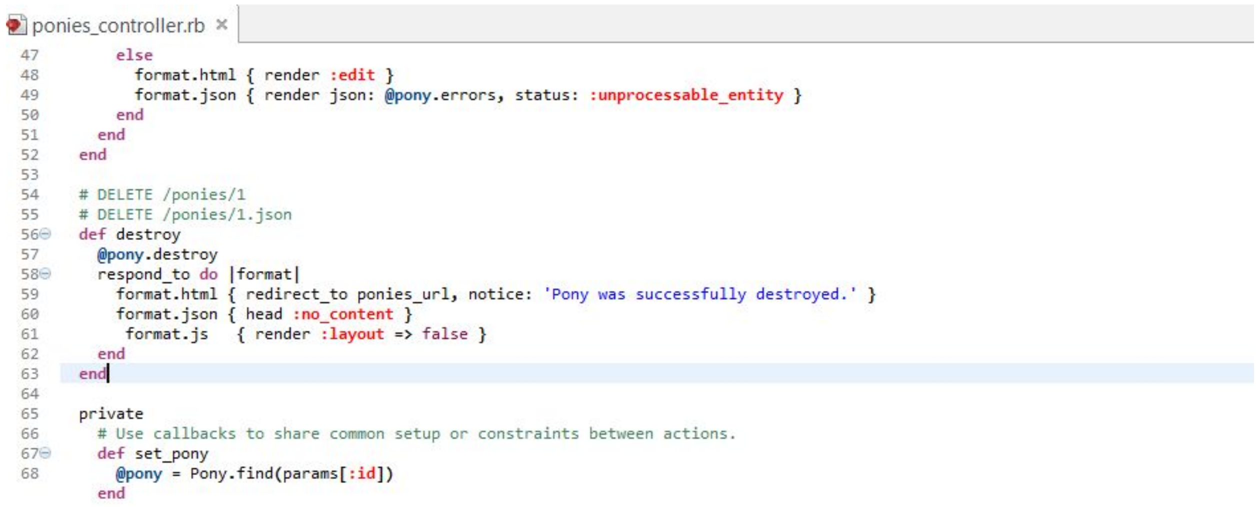
```
$('.delete_pony').bind('ajax:success', function() {
  $(this).closest('tr').fadeOut();
});
```

Now Open your controller file which is placed at app/controllers/ponies_controller.rb and add the following code in destroy method as shown below –

```
# DELETE /ponies/1
# DELETE /ponies/1.json
def destroy
  @pony = Pony.find(params[:id])
  @pony.destroy
  respond_to do |format|
    format.html { redirect_to ponies_url }
    format.json { head :no_content }
    format.js   { render :layout => false }
  end

end
```

At finally controller page is as shown image.



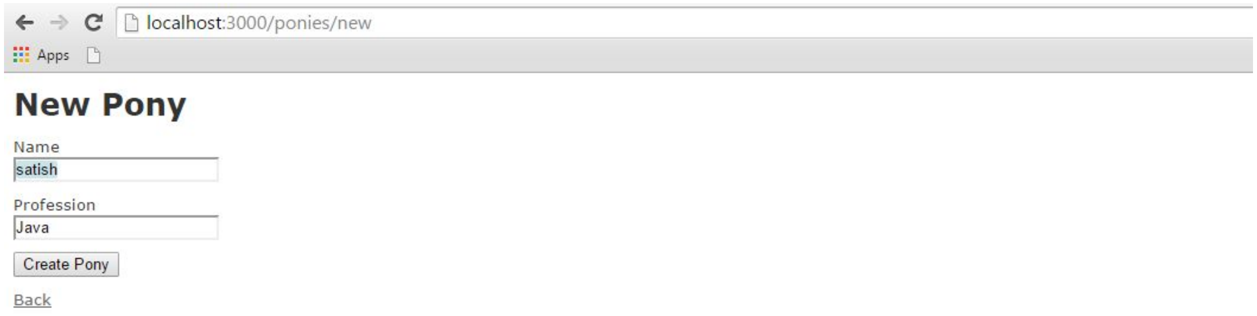
ponies_controller.rb

```
      else
        format.html { render :edit }
        format.json { render json: @pony.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /ponies/1
  # DELETE /ponies/1.json
  def destroy
    @pony.destroy
    respond_to do |format|
      format.html { redirect_to ponies_url, notice: 'Pony was successfully destroyed.' }
      format.json { head :no_content }
       format.js   { render :layout => false }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_pony
      @pony = Pony.find(params[:id])
    end
```

Now run an application, Output called from http://localhost:3000/ponies/new, it will looks like as following image



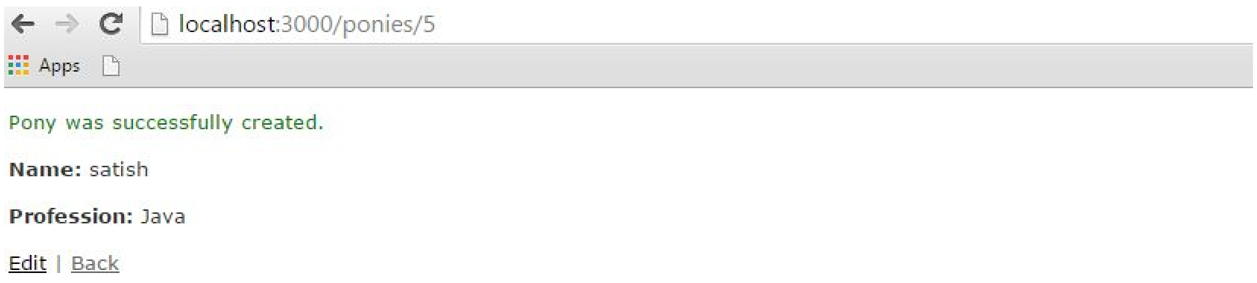
localhost:3000/ponies/new

New Pony

Name: satish

Profession: Java

Press on create pony button, it will generate the result as follows



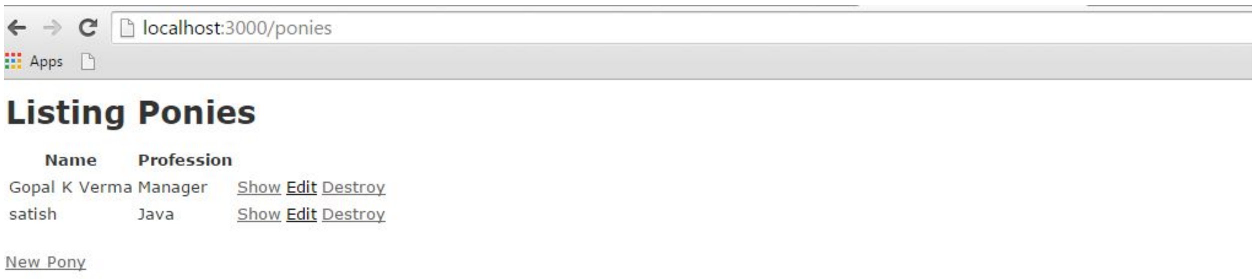
Pony was successfully created.

Name: satish

Profession: Java

Edit | Back

Now click on back button, it will show all pony created information as shown image

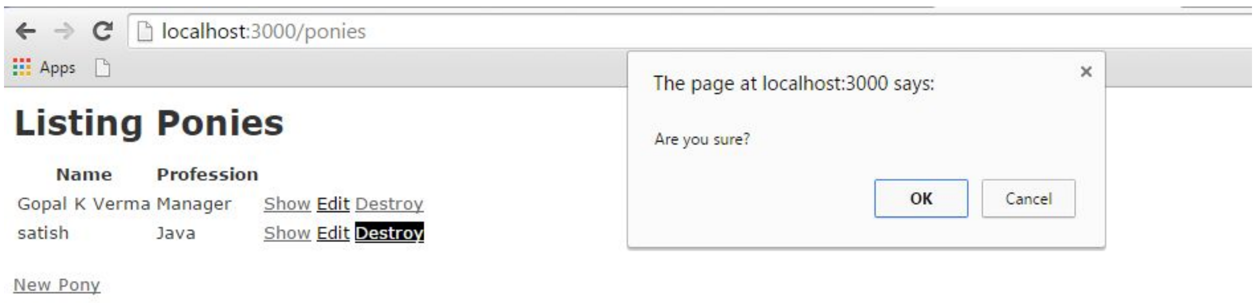


Listing Ponies

Name	Profession			
Gopal K Verma	Manager	Show	Edit	Destroy
satish	Java	Show	Edit	Destroy

New Pony

Till now, we are working on scaffold, now click on destroy button, it will call a pop-up as shown below image, the pop-up works based on Ajax.


localhost:3000/ponies

The page at localhost:3000 says:

Are you sure?

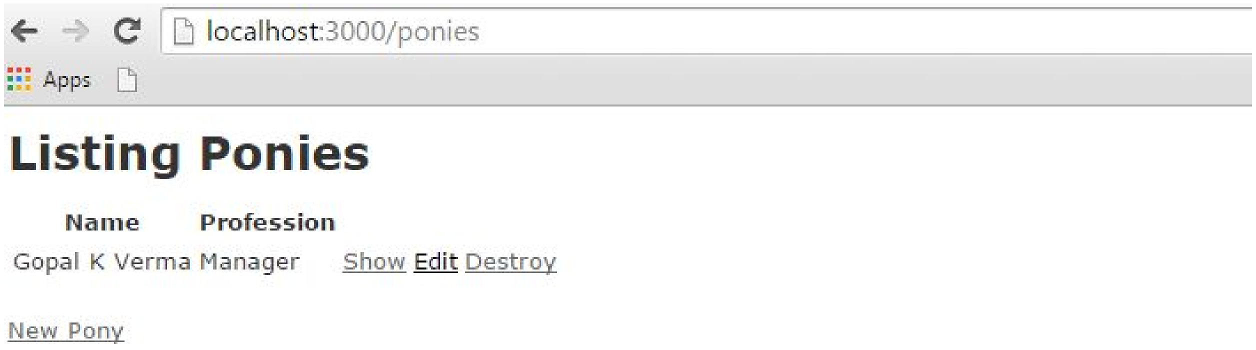
OK Cancel

Listing Ponies

Name	Profession	
Gopal K Verma	Manager	Show Edit Destroy
satish	Java	Show Edit Destroy

New Pony

If Click on ok button, it will delete the record from pony. Here I have clicked ok button. Final output will be as follows –


localhost:3000/ponies

Listing Ponies

Name	Profession	
Gopal K Verma	Manager	Show Edit Destroy

New Pony