Blog application revisited

Before using any gems or ready to run code in your application, we need a good understanding of "ruby programming" and "rails way" managing applications. In the coming couple of lectures our focus will be on

- Rails structure for managing application
- Ruby Programming

So lets first throughly understand how the rails develop application using scaffold or other gems. To achieve this milestone we build an application without scaffold to under the rails way.

Creating a blog application

Create a new blog application

```
$ cd railsApp
$ rails new blog -d postgresql
$ cd blog
```

Say "Hello", Rails

Create a new controller

```
$ rails generate controller welcome index
```

Open the app/views/welcome/index.html.erb file in your editor

app/view/welcome/index.html.erb

```
<h1>Hello, Rails!</h1>
```

Testing app in browser

Let's test the blog application in the browser. Fire up the browser and type and wait do not forget to create the database first.

```
$ rake db:create
```

and then type in the browser to test the app

```
$ localhost:3000
$ localhost:3000/welcome/index
```

Setting the application home page

config/routes.rb

```
get 'welcome/index'
root 'welcome#index'
```

Getting up and routing

Entering routes to your resources

config/routes.rb

```
Rails.application.routes.draw do
  resources :articles

  root 'welcome#index'
end
```

If you run rake routes, you'll see that it has defined routes for all the standard RESTful actions.

```
$ rake routes
```

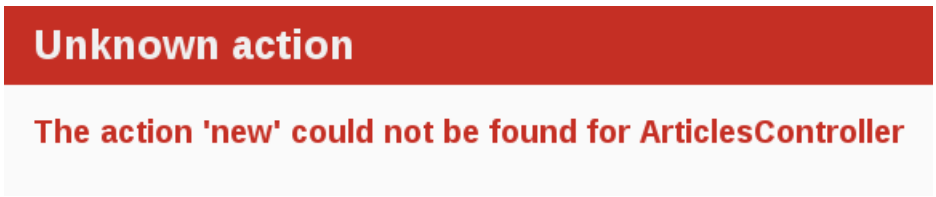
Laying down the ground work

Firstly, you need a place within the application to create a new article. If you type *http://localhost:3000/articles/new* you will receive

The routes need a controller defined to serve the request.

```
$ rails g controller articles
```

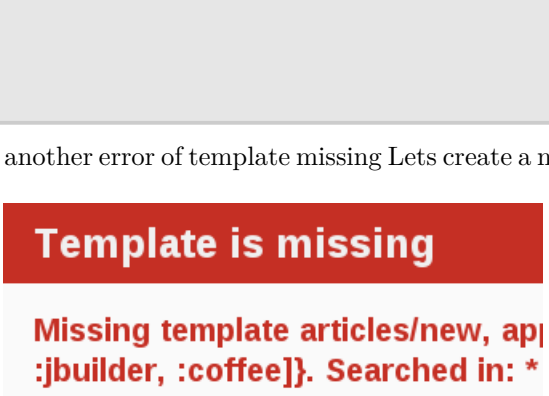
If you refresh the page now you will receive a new error Now define an action for the new page



app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController
  def new
  end
end
```

When you refresh again you will receive another error of template missing Lets create a new view file **app/view/article/new.html.er**



and write this code

```
<h1> New Article </h1>
```

The first form

We need a form builder to create a form in template

app/views/articles/new.html.erb

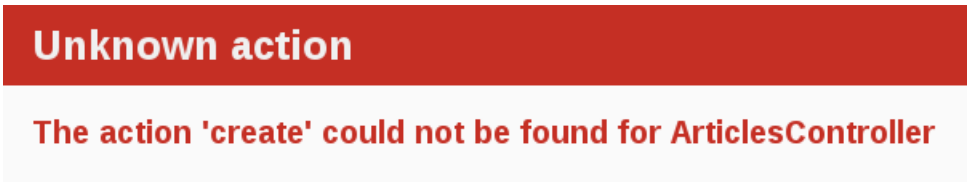
```
<%= form_for :article do |f| %>
  <p>
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </p>

  <p>
    <%= f.label :body %><br>
    <%= f.text_area :body%>
  </p>

  <p>
    <%= f.submit %>
  </p>
<% end %>
```

If you refresh the page you will see a form. There is one problem with this form. When you check its source code from browser, you will see it is going to the same page that you are at the moment now.

The form needs to use a different URL in order to go somewhere else. this can be done quite simply with :url option of form_for.

```
<%= form_for :article, url: articles_path do |f| %>
```

Check the routes, with articles we two methods GET and POST. Rails will automatically call the POST method.



Creating articles

Lets now define create action

app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController
  def new
  end

  def create
  end
end
```

Now if you refresh you will receive another familiar error template missing. To remove the error lets add code to the Article controller

```
def create
  render plain: params[:article].inspect
end
```

The render method here is taking a very simple hash with a key of plain and value of params[:article].inspect. The params method is the object which represents the parameters (or fields) coming in from the form.

This form is not saving the articles yet. As you have no database on the back end.

Creating the Articles model

Model in Rails use a singular name, and their corresponding database tables use a plural name. Rails provides a generator for creating models.

```
$ rails generate model Article title:string body:text
```

The rails migration file

```
class CreateArticles < ActiveRecord::Migration
  def change
    create_table :articles do |t|
      t.string :title
      t.text :body

      t.timestamps null: false
    end
  end
end
```

Let create and migrate the database

```
$ rake db:migrate
```

Saving data in the controller

Back in ArticlesController, we need to change the create action to use the new Article model to save the data in the database.

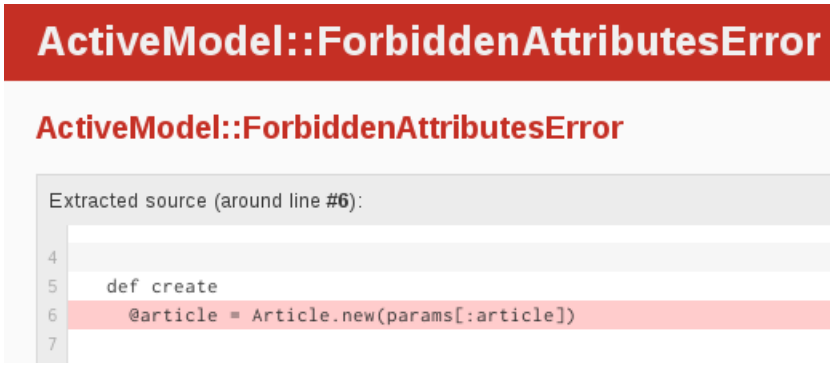
app/controllers/articles_controller.rb

```
def create
  @article = Article.new(params[:article])

  @article.save
  redirect_to @article
end
```

Now you try to save the article, we will receive another error. This one is called strong parameters, which requires us to tell Rails exactly which parameters are allowed into our controller actions.

```
@article = Article.new(params.require(:article).permit(:title, :body))
```

This is often factored out into its own method so it can be reused by multiple actions in the same controller, for example create and update.




```
def create
  @article = Article.new(article_params)

  @article.save
  redirect_to @article
end

private
  def article_params
    params.require(:article).permit(:title, :body)
  end
```

Showing Articles

The special syntax :id tells rails that this route expects an :id parameter, which in our case will be the id of the article.

```
class ArticlesController < ApplicationController
  def show
    @article = Article.find(params[:id])
  end
```

Now, create a new file app/views/articles/show.html.erb with the following content:

app/views/articles/show.html.erb

```
<p>
  <strong>Title:</strong>
  <%= @article.title %>
</p>

<p>
  <strong>Text:</strong>
  <%= @article.body %>
</p>
```

Listing all articles

We still need a way to list all our articles, so let's do that. The route for this as per output of rake routes is:

app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.all
  end

  def show
    @article = Article.find(params[:id])
  end

  def new
  end
```

And then finally, add the view for this action, located at app/views/articles/index.html.erb:

app/views/articles/index.html.erb

```
<h1>Listing articles</h1>

<table>
  <tr>
    <th>Title</th>
    <th>Text</th>
  </tr>

  <% @articles.each do |article| %>
    <tr>
      <td><%= article.title %></td>
      <td><%= article.body %></td>
    </tr>
  <% end %>
</table>
```

Adding Links

Now we have create, show, and list articles. Now let's add some links to navigate through pages. Open the page app/views/welcome/ind

app/views/welcome/index.html.erb

```
<h1>Hello, Rails!</h1>
<%= link_to 'My Blog', controller: 'articles' %>
```

Adding a link to new article in articles index page.

app/views/articles/index.html.erb

```
<%= link_to 'New article', new_article_path %>
```

Now, add another link in app/views/articles/new.html.erb, underneath the form, to go back to the index action:

app/views/articles/new.html.erb

```
<%= form_for :article, url: articles_path do |f| %>
...
<% end %>

<%= link_to 'Back', articles_path %>
```

Finally, add a link to the app/views/articles/show.html.erb template to go back to the index action as well

app/views/articles/show.html.erb

```
<p>
  <strong>Title:</strong>
  <%= @article.title %>
</p>

<p>
  <strong>Body:</strong>
  <%= @article.body %>
</p>

<%= link_to 'Back', articles_path %>
```

Adding some validation

Lets add some validation rule to the app/models/article.rb

app/models/article.rb

```
class Article < ActiveRecord::Base
  validates :title, presence: true,
                    length: { minimum: 5 }

end
```

After adding validation, the @article.save returns false, so we need to catch the error and keep the user on the same form.

app/conrollers/articles_conrollter.rb

```
def new
  @article = Article.new
end

def create
  @article = Article.new(article_params)

  if @article.save
    redirect_to @article
  else
    render 'new'
  end
end
```

If you reload http://localhost:3000/articles/new and try to save an article without a title, Rails will send you back to the form, but that's not very useful. You need to tell the user that something went wrong. To do that, you'll modify app/views/articles/new.html.erb to check for error messages:

app/views/articles/new.html.erb

```
<%= form_for :article, url: articles_path do |f| %>

<% if @article.errors.any? %>
  <div id="error_explanation">
    <h2>
      <%= pluralize(@article.errors.count, "error") %> prohibited
      this article from being saved:
    </h2>
    <ul>
      <% @article.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>

<p>
  <%= f.label :title %><br>
  <%= f.text_field :title %>
</p>

<p>
  <%= f.label :body %><br>
  <%= f.text_area :body %>
</p>

<p>
  <%= f.submit %>
</p>

<% end %>
<%= link_to 'Back', articles_path %>
```

Updating Articles

app/conrollers/articles_conrollter.rb

```
def edit
  @article = Article.find(params[:id])
end
```

Lets now create the edit.html.erb file template

app/views/articles/edit.html.erb

```
<h1>Editing article</h1>

<%= form_for :article, url: article_path(@article), method: :patch do |f| %>

<% if @article.errors.any? %>
  <div id="error_explanation">
    <h2>
      <%= pluralize(@article.errors.count, "error") %> prohibited
      this article from being saved:
    </h2>
    <ul>
      <% @article.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>

<p>
  <%= f.label :title %><br>
  <%= f.text_field :title %>
</p>

<p>
  <%= f.label :body %><br>
  <%= f.text_area :body %>
</p>

<p>
  <%= f.submit %>
</p>

<% end %>
<%= link_to 'Back', articles_path %>
```

Now add an update action to the controller

app/controllers/articles_controller.rb

```
def update
  @article = Article.find(params[:id])

  if @article.update(article_params)
    redirect_to @article
  else
    render 'edit'
  end
end
```

Now we need an edit link to modify the record in index file.

app/views/articles/index.html.erb

```
<table>
<tr>
  <th>Title</th>
  <th>Text</th>
  <th colspan="2"></th>
</tr>

<% @articles.each do |article| %>
  <tr>
    <td><%= article.title %></td>
    <td><%= article.body %></td>
    <td><%= link_to 'Show', article_path(article) %></td>
    <td><%= link_to 'Edit', edit_article_path(article) %></td>
  </tr>
<% end %>
</table>
```

Using partials to clean up duplication

The edit page looks very similar to the new page, they share the same code to display form. We can remove the duplication using a view partials.

app/views/articles/_form.html.erb

```
<%= form_for @article do |f| %>

  <% if @article.errors.any? %>
    <div id="error_explanation">
      <h2>
        <%= pluralize(@article.errors.count, "error") %> prohibited
        this article from being saved:
      </h2>
      <ul>
        <% @article.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <p>
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </p>

  <p>
    <%= f.label :body %><br>
    <%= f.text_area :body %>
  </p>

  <p>
    <%= f.submit %>
  </p>

<% end %>
```

Now update the new.html.erb and edit.html.erb to connect the form partials.

app/views/articles/new.html.erb

```
<h1>New article</h1>

<%= render 'form' %>

<%= link_to 'Back', articles_path %>
```

app/views/articles/edit.html.erb

```
<h1>Edit article</h1>

<%= render 'form' %>

<%= link_to 'Back', articles_path %>
```

0.0.1 Deleting articles

We're now ready to cover the "D" part of CRUD (Create, Read, Update, Delete), deleting articles from the database. Following the REST convention, the route for deleting articles as per output of rake routes is:

Rake routes

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

Add the destroy action to the controller

app/controllers/articles_controller.rb

```
def destroy
  @article = Article.find(params[:id])
  @article.destroy

  redirect_to articles_path
end
```

The full view of the articles controller code.

app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.all
  end

  def show
    @article = Article.find(params[:id])
  end

  def new
    @article = Article.new
  end

  def edit
    @article = Article.find(params[:id])
  end

  def create
    @article = Article.new(article_params)

    if @article.save
      redirect_to @article
    else
      render 'new'
    end
  end

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

    if @article.update(article_params)
      redirect_to @article
    else
      render 'edit'
    end
  end

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

    redirect_to articles_path
  end

  private
    def article_params
      params.require(:article).permit(:title, :body)
    end
end
```

Add the destroy link to the listing of articles in app/views/articles/index.html.erb.

app/views/articles/index.html.erb

```
      <td><%= link_to 'Destroy', article_path(article),
             method: :delete,
             data: { confirm: 'Are you sure?' } %></td>
```

Adding a second model

```
$ rails generate model Comment commenter:string comment:text article:references
```

The references command automatically adds the association in the model.
Do the following

1. Generate controller for the comments and its associated views.
2. Apply validations.