

Full Stack Notes

Intro to Rails - Building a Blog

In this Rails tutorial we will create a simple blogging application. This application will allow the users to:

- Post new blog entries (title, body, timestamp).
- Edit or delete existing blog entries.
- View all blog entries in reverse-chronological order.
- On post, the system will ensure the user has provided a title.

This module is a simplified version of the [Getting Started with Rails](#) guide.

Table of Contents

- 1 [Objectives](#)
- 2 [Setup Instructions](#)
- 3 [Starting the Server](#)
- 4 [Scaffolding](#)
- 5 [Database Creation & Migration](#)

- 6 [Testing the Scaffolding](#)
- 7 [MVC Bundles](#)
- 8 [RESTful Actions and Routes](#)
- 9 [Adding Validation](#)
- 10 [Creating the Blog View](#)
- 11 [Layouts](#)
- 12 [Script Console](#)

Objectives

Upon completion of this tutorial, you should be able to:

- Create a new Rails project from the command line.
- Start and stop the testing web-server.
- Generate a scaffold for a particular MVC bundle.
- Prepare your project database using Rake.
- Edit and modify a view associated with a controller action.
- Add simple validation to a model.
- Interact with a model via the console.

*MVC = Model View Controller

Setup Instructions

Let's create a new Rails project named 'blog'.

- In your WSL terminal type: `rails new blog`

This command will generate a 'blog' folder along with a number of sub-folders and files. Your Rails application 'exists' within these folders and files.

Be sure to run this command from inside the folder where you wish to place your application folder.

Starting the Server

Now we'll run the built-in test web-server:

- Open a second console window, navigate to your 'blog' folder and type:

```
rails s
```

- Load the default welcome screen for your application here:

<http://localhost:3000>

The command you ran above tells Ruby to launch an HTTP Web Server through which you can test your Rails application. The name of the server you launched is Puma.

RESOURCES

- [Puma Web Server](#)

Scaffolding

Let's tell Rails to scaffold us up an MVC-bundle for a blog post:

- From the command line type: `rails generate scaffold Post title:string body:text`

Here we've specified that we wish to generate a scaffold named 'Post'. We have further specified that each post contains a 'title' (which is a string) and a 'body' (which is a text blob).

We'll see later on that by Rails will also add a 'created_at' and 'updated_at' property to each post.

The scaffold generator takes the name of the model (the resource) and optionally a list of field names and types.

Files generated as a result of the scaffolding:

- `\blog\app\controllers\posts_controller.rb`
- `\blog\app\views\posts\index.html.erb`
- `\blog\app\views\posts\show.html.erb`
- `\blog\app\views\posts\new.html.erb`
- `\blog\app\views\posts\edit.html.erb`

- `\blog\app\views\posts_form.html.erb`
- `\blog\app\models\post.rb`
- `\blog\app\helpers\posts_helper.rb`
- `\blog\db\migrate\001_create_posts.rb`

Amongst others...


Database Creation & Migration

The default Rails database is SQLite. Let's create the database table where we will store our blog posts:

- From the command line type: `rails db:migrate`

This command creates a new 'posts' table in your database. This table will contain five columns:

- id (Auto-incrementing integer index)
- title (string)
- body (text blob)
- created_at (timestamp created automatically)
- updated_at (timestamp created automatically)

 **Note:** Even though we created a scaffold for 'Post' (singular) the table created by Rake was named 'posts' (plural). This is a Rails standard. The controller and the views will also reference 'posts' (plural).

If you want to better understand the DB Migration you should investigate the following file from within your blog project folder:

```
blog\db\migrate\001_create_posts.rb
```

(In the place of 001 in the filename will actually be a long timestamp.)

This file was created when you ran the scaffolding command. It details (in Ruby) the database table associated with the scaffolded resource.

RESOURCES

- [Rails DB Migrations](#) (Rails Guides)
- [SQLite @ Wikipedia](#)
- [SQLite.org](#)

Testing the Scaffolding

We should now be able to load our posts controller:

<http://127.0.0.1:3000/posts>

We can even add a new post by click on the 'New post' link.

MVC Bundles

When we created the `Post` scaffolding, Rails created a `posts` controller with a number of actions to allow for [CRUD](#) tasks within our `posts` table in the database.

For example, the `index` action fetches all posts from the model. This action is accessible via the following url:

```
http://127.0.0.1:3000/posts/
```

This index action has an associated view to display the posts:

```
blog\app\views\posts\index.html.erb
```

The controller uses the `Post` model to access the `posts` table in the database:

```
blog\app\models\post.rb
```

RESOURCES

- [Rails Controllers Overview](#) (Rails Guides)

RESTful Actions and Routes

All controller methods represent actions that are accessible via a URL. Each action has an associated view.

The scaffolding creates an `index`, `show`, `new`, `edit`, `create`, `update` and `destroy` action along with associated

routes. These are called the seven RESTful actions/routes.

For example, the 'new post' action is found in the `new` method of the `posts` controller.

The `posts` controller can be found here:

```
blog\app\controllers\posts_controller.rb
```

Its `new` action is accessible via the following URL:

```
http://127.0.0.1:3000/posts/new
```

The associated view is located here:

```
blog\app\views\posts\new.html.erb
```

RESOURCES

- [What is REST](#)

Adding Validation

Let's add some simple validation to our blog posting.

- Open the `Post` model located here: `blog\app\models\post.rb`

The `Post` class defined in this file allows us to interact with our `posts` table in the database. It is currently empty, but it inherits the database functionality from the `ActiveRecord::Base` class (by way of the `ApplicationRecord` class).

- Add the following line inside the `Post` class:

```
validates :title, presence: true
```

Now, before a post can be added to the `posts` table Rails will ensure that the user has specified a title. Try adding [a new post](#) without a title.

RESOURCES

- [Active Record Validations](#) (Rails Guides)

Creating the Blog View

Let's edit the `posts` controller's `index` view to make this look more like a real blog. Open the following file:

```
blog\app\view\posts\index.html.erb
```

Ruby code is embedded into this view within `<%` and `%>` tags. If we want to print a string from Ruby into the HTML we use `<%=` and `%>`.

The `@posts` variable was 'given' to our view by the `index` action of the `posts` controller. Notice that we are using a for-in loop to display all posts. The view will be more 'blog-like' if the posts are shown in reverse

chronological order.

Replace the current `index` view with the following, and study this code until you can sing it blindfolded. Notice how the post `title` and `body` are accessed. What does the `link_to` method do?

```
<h1>Our Blog</h1>
<br />
<% @posts.each do |post| %>
  <h2> <%= post.title %> </h2>
  <p> <%= post.body %> </p>
  <p>
    Created at:
    <%= link_to post.created_at, post %>
    - <%= link_to 'Edit', edit_post_path(post) %>
    - <%= link_to 'Destroy', post, :confirm => 'Are you sure?', :method => :delete %>
  </p>
  <hr />
<% end %>
<br />
<%= link_to 'New post', new_post_path %>
```

To ensure that blog posts appear in reverse-chronological order, edit the `index` action of the `posts` controller. Modify the line where the `@posts` variable is created to look like this:

```
@post = Post.order("created_at DESC")
```

RESOURCES

- [Erb - Ruby Embedded in HTML](#)

Layouts

You may have noticed that the views associated with controller actions are not complete HTML files. This is because each view is inserted inside an HTML layout shell before it is displayed. The layout for your application can be found here:

```
\blog\app\views\layouts\application.html.erb
```

The each view is inserted into a layout before the HTML is sent to the user's browser. The view is inserted where the layout reads:

```
<%= yield %>
```

The layout yields to a particular view; how very polite, proper and posh! Try putting some extra marking in the layout and then navigating around your app.

RESOURCES

- [Rails Layout and Rendering](#) (Rails Guides)

Script Console

The Rails console allows us to interact with our project models.

- From a command prompt type: `rails c`

(Ensure that you are in your project's root folder.)

From the console try playing around:

```
p = Post.all # Ask the Post model for all posts.  
p[0].title  
first = Post.first  
new_post = Post.new title: 'Hello from Console', body: 'This is a test'  
new_post.save # Save our new post to the database.  
Post.all
```

A second way to add from the console:

```
Post.create title: 'Another Console Post', body: 'Second console body text.'
```

