Full Stack Notes

View Helpers

View helpers are method we can call in our ERB views to generate markup. In the previous section we looked at the link_to helper.

There are many other helpful helpers.

Table of Contents

Displaying Images

The image_tag helper builds an HTML `<img />` tag for the specified file. By default, files are loaded from the `app/assets/images` folder.

To display the `app/assets/images/fish.png` image:

```
<%= image_tag "fish.png" %>
```

We can also any of the `img` tag attributes using symbols:

```
<%= image_tag "fish.png", , :width ='20%' %>
```

This would generate:

```
<img alt="Fish" src="/assets/fish.png" width="20%" />
```

For model objects that include an image URL string property, or an ActiveStorage attached image, we use the helper like this:

```
<%= image_tag @product.image %>
```

Assuming that `@product` is an ActiveRecord model object.

Rails View Partials

Partials are a device for breaking the view rendering process into more manageable chunks. With a partial, you can move the code for rendering a particular view to its own file. This allows us to keep our view code DRY.

Render the `_menu.html.erb` file found in the current folder:

```
<%= render 'menu' %>
```

This is a short-cut for:

```
<%= render partial: 'menu' %>
```

If the `_menu.html.erb` file isn't in the current folder, for example the `app/view/shared` folder:

```
<%= render 'shared/menu' %>
```

NOTE: Partial filenames must always begin with an underscore, but we don't use the underscores in the `render` arguments.

Rendering Partials for Objects or Collections

Given a @product variable that contains a Product object render a `_product.html.erb` file found in the products view folder.

```
<%= render @product %>
```

Given a collection of Product objects in @products render the `views/products/_product.html.erb` once for each Product.

```
<%= render @products %>
```

These are shortened forms for the fully expanded calls to the render command. The first of these is equivalent to:

```
<%= render partial: 'product', object: @product %>
```

The second is equivalent to:

```
<%= render partial: 'product', collection: @products %>
```

RESOURCES

Rails partials have a lot more to offer, including local variables. Be sure to read through:

- View Partials from the Layouts and Rendering Rails Guide.

Working With Model Forms

We can use the `form_with` helper to bind a form to a model object.

Imagine a controller where the `new` and `edit` actions load a `Post` object into a `@post` variable. In the view, we could build a form bound to the `Post` model object:

```
<%= form_with(model: @post, local:true) do |f| %>
  <div>
      <%= f.label :title %><br/>
      <%= f.text_field :title %>
  </div>
  <div>
    <%= f.label :body %><br/>
    <%= f.text_area :body %>
  </div>
  <div>
    <%= f.label :image %><br/>
    <%= f.text_field :image %>
  </div>
```

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

The `f` block argument is a form builder. We use it to build the various components of our form.

For example, this code:

```
<%= f.label :image %>
<%= f.text_field :image %>
```

Would generate these html form tags, associated with the `Post` object's `image` property.

```
<label for="post_image">Image</label>
<input id="post_image" name="post[image]" size="30" type="text" value="" />
```

RESOURCES

More information on dealing with generic forms, or model-centric forms read the Form Helper Rails Guide.

Model Forms with Associations

Rails makes it easy to associate one model with another by way of a foreign key. In our CRM example the customer model had a `province_id` property. To link the two models:

in `app\models\customer.rb`:

```
belongs_to :province
```

in `app\models\province.rb`:

```
has_many :customers
```

Then within the customers `_form` partial:

```
<div class="field">
  <%= f.label :province_id %><br />
  <%= f.collection_select :province_id, @provinces, :id, :name %>
</div>
```

This view code assumes that the `@provinces` instance variable contains a collection of all the Province objects from the provinces table. This instance variable would have to be added to any controller actions using this `_form` partial.

It also assumes that Province objects have a `name` property containing the name of the province.

Select Dropdowns with form_tag

The `f.collection_select` helper works nicely with `form_for`, but sometimes you want to build a select dropbox for use with a `form_tag`. For this we use a combination of `select_tag` and `options_for_select`.

```
<%= form_tag some_route_path do %>
   <%= select_tag :my_option, options_for_select(@some_array) %>
   <%= submit_tag "Search" %>
<% end %>
```

When this form is submitted the option selected by the user in the select dropdown will be found in `params[:my_option]`.

The `@some_array` variable should be an array of strings to use as options for the select dropdown.

If your options are coming from an ActiveRecord collection, you can use the `options_from_collection_for_select` helper. The added benefit of this helper is that the `value` attributes of each `option` tag can be made to contain the ActiveRecord `id` of the collection members.

So for example, if you had an `@categories` collection of `Category` objects where each category had a `name`:

```
<%= select_tag :category, options_from_collection_for_select(@categories, 'id', 'name') %>
```

On submission of the form the selected category id could be found in `params[:category]`.

RESOURCES

- Rails API: options_for_select
- Rails API: options_from_collection_for_select
- Rails Guide: The Select and Option Tags
