Date: 2/10/19

Assumptions

Setup

None required.

Morning Session

Updating the Controller of a Nested Project

Let's remind ourselves about the routes we have setup for the Medium project:

```
Rails.application.routes.draw do
  resources :articles do
    resources :comments
    resources :authors
  end
  resources :authors
end
```

First up we need to update the Controller of the nested resources:

```
class CommentsController < ApplicationController
  before_action :set_comment, only: [:show, :edit, :update, :destroy]
  before_action :set_article
```

Because we always have the Comments after the selection of an Article, regardless of the action being checked, we set the article to an instance variable as the article identifier is included in the params hash.

```
  private
    # Use callbacks to share common setup or constraints between actions.
    def set_article
      @article = Article.find( params[ :article_id ] )
    end
```

When Rails sets up the scaffold for the Comments, it setup the initial route paths before we changed the routes.rb. So we need to update the paths to those listed in our routes, those shown in rails/info/routes or rake routes.

```
  # POST /comments
  # POST /comments.json
  def create
    @comment = Comment.new(comment_params)
    @comment.article_id = @article.id
```

```
    respond_to do |format|
      if @comment.save
        format.html { redirect_to article_comment_path( @article, @comment
), notice: 'Comment was successfully created.' }
        format.json { render :show, status: :created, location: @comment }
      else
        format.html { render :new }
        format.json { render json: @comment.errors, status:
:unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /comments/1
  # PATCH/PUT /comments/1.json
  def update
    respond_to do |format|
      if @comment.update(comment_params)
        format.html { redirect_to article_comment_path( @article, @comment
), notice: 'Comment was successfully updated.' }
        format.json { render :show, status: :ok, location: @comment }
      else
        format.html { render :edit }
        format.json { render json: @comment.errors, status:
:unprocessable_entity }
      end
    end
  end

  # DELETE /comments/1
  # DELETE /comments/1.json
  def destroy
    @comment.destroy
    respond_to do |format|
      format.html { redirect_to article_comments_path( @article ), notice:
'Comment was successfully destroyed.' }
      format.json { head :no_content }
    end
  end
```

The upshot is replacing the paths that utilise the symbols and instance variables as shorthand with actual paths. We also have to ensure that the parent identifiers are included in the create method.
Within the controller, we always use the instance variables.

Updating the Views of a Nested Project

app/views/comments/index.html.erb:

```
<p id="notice"><%= notice %></p>

<h1>Comments</h1>
```

```
<table>
  <thead>
    <tr>
      <th>Response</th>
      <th>Article</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @comments.each do |comment| %>
      <tr>
        <td><%= comment.response %></td>
        <td><%= comment.article.title %></td>
        <td><%= link_to 'Show', article_comment_path(@article, comment)
%></td>
        <td><%= link_to 'Edit', edit_article_comment_path(@article, comment)
%></td>
        <td><%= link_to 'Destroy', article_comment_path(@article, comment),
method: :delete, data: { confirm: 'Are you sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New Comment', new_article_comment_path %>
```

We can utilise the article's **title** directly as we have setup the comment as belonging to the article in the models.
Thereafter, we update the paths appropriately as we did in the controller. Please note that the article is always used as the instance variable we created in the controller.

app/views/comments/show.html.erb

```
<p id="notice"><%= notice %></p>

<p>
  <strong>Response:</strong>
  <%= @comment.response %>
</p>

<p>
  <strong>Article:</strong>
  <%= @comment.article.title %>
</p>

<%= link_to 'Edit', edit_article_comment_path( @article, @comment ) %>
<%= link_to 'Back', article_comments_path( @article ) %>
```

We can update the show to access the Title of the article relating to the comment being shown. We have to update the route paths again; the parameters to the paths can be specified but they are optional, probably as we are not changing the id params.

app/views/comments/edit.html.erb

```
<h1>Editing Comment</h1>

<%= render 'form', article: @article, comment: @comment %>

<%= link_to 'Show', article_comment_path(@article, @comment) %> |
<%= link_to 'Back', article_comments_path( @article ) %>
```

For the scripts accessing the partial form, we need to pass in both the article and the comment. We also have to update the paths again as in the other scripts.

app/views/comments/new.html.erb

```
<h1>New Comment</h1>

<%= render 'form', article: @article, comment: @comment %>

<%= link_to 'Back', article_comments_path( @article ) %>
```

The new script follows the same pattern as the edit script; again the parameter to the path is optional.

app/views/comments/_form.html.erb

```
<%= form_with(model: [@article, comment], local: true) do |form| %>
  <% if comment.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(comment.errors.count, "error") %> prohibited this
comment from being saved:</h2>

      <ul>
        <% comment.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= form.label :response %>
    <%= form.text_area :response %>
  </div>
<!--
  <div class="field">
    <%= form.label :article_id %>
    <%= form.text_field :article_id %>
```

```
  </div>
-->
  <div class="actions">
    <%= form.submit %>
  </div>
<% end %>
```

The parameters are passed; the **@article** is a reference to the instance variable but the **comment** (the nested resource) is specifically not as we never want to be updating the instance variable within an edit.
I've commented out the editing of the associated article as it is not particularly appropriate to change it at this point.
Important: The order of the parameters passed to **model** in the partial form creates the name of the path that Rails uses under the covers, so the order is important to ensure that order specified is that used as a path in the routes.

Also Important: Be very careful with the advice on forums and places like Stack Overflow et al on these as you will get a lot of variance in the answers and some are particularly incorrect, even though it will look like they are working.