COMM049: Mobile Application Development

Week 9 – Adding Comments to Articles Autumn 2016

Commenting in Articles

I want to focus on associations between classes for the moment so will defer the authentication until later if we have time.

We are going to see some magic happen.

Exercise:

I would like you to generate a scaffold for a Comment model. Comment needs to have attributes: article_id of type integer; details of type text; and, author of type string. Refer to last week's exercise for the details of how to do this.

Once the scaffold has been generated, you need to modify the Article and Comment class definitions to set up a one-to-many association between events and comments. Within Rails, this is really very easy. You need to make small changes to your Article and Comment class definitions.

Firstly, insert this line into your Article class definition:

has_many :comments

Secondly, insert this line into your Comment class definition:

belongs_to :article

You might want to make one further enhancement to this. You will remember that the respective scaffolds enable us to delete both Articles and Comments. However, if we delete an Article, we don't really want its associated Comments cluttering up the application as "orphaned dependents". Instead, we will ensure that any unwanted dependents are simply destroyed when we delete an Article. We do this by modifying the line we added into the Article class by adding the text in bold:

has_many :comments**, :dependent => :destroy**

Now for the magic. Make sure you have used the form to create at least one Article in your application. You should be able to work out how to get to the index page for comments by now. Go there and create a new comment, making sure that the article_id is set to 1 (assuming you have not deleted the first article you created).

Now, open a new terminal/command window and go to the root folder of your project. Enter the following:

$ rails console

This will open an irb that is attached to your project. This means the irb has access to all the objects that are executing.

You can create an array of all the current events thus:

> articles = Article.all

After that, you can pick out the first blog article:

> first_article = articles[0]

and then finally you can see all the comments that are associated with it:

> first_article.comments

You can even pick out the first comment:

> first_article.comments[0]

Simple Exercise:

Add some more articles and comments, and use the rails console to check all the expected associations are in place.

Hiding the foreign key

Having to add in the foreign key by hand is just plain silly, so lets fix that.

As always, we will do this bit by bit. The first thing to do is to add a link when we are showing an article to enable someone to add a new comment.

To do that, you need to add the following line to apps/view/articles/show.html.erb

Underneath the link to "Edit":

<%= link_to 'New Comment', new_comment_path %> |

Try this out. You will see that at least we can comment on the articles we are interested in, but we still need to add the foreign key ourselves.

We fix this by passing a key:value pair with the foreign key we want.

Prior to Rails 3.2, we would have had to do this:

<%= link_to 'New Comment', new_comment_path(:article_id => @article.id) %> |

However, this pattern occurs many times in any one application so this has been simplified. Firstly, the assumption is that when you pass a reference to an object as a parameter, all you really mean is to pass the id for that object. So the value on the right hand side can be simplified to "@article" (you will see this in the edit link above).

Secondly the ":symbol =>" pattern is so common that rails core decided to simplify this to "symbol: ". So, what you actually need to do is to change the above line to:

<%= link_to 'New Comment', new_comment_path(article_id: @article) %> |

It is a little more cryptic, but once you know where it comes from it is quite easy (I hope!) to remember that this is a short hand for a hash.

Now we need to pick up this parameter in the CommentsController. We need to do a similar thing – assign the @article_id instance variable of a comment the

corresponding value of the parameter that has been forwarded to this controller. Edit the first line of the "new" action so that it looks like this:

@comment = Comment.new(article_id: params[:article_id])

Try it now. Go to an article and click the link to create a new comment. The corresponding article_id should appear in the first line of the form for creating a new Comment. (if it doesn't then the first thing to try is to close the server and restart it.)

But that is still not quite where we want to be. We want to replace the line of the form where we add the foreign key with a "hidden field" that simply passes our parameter in to the creation of a new Comment object.

The first thing to do is to delete the associated with the entry of "article_id", in app/views/comments/_form.html.erb

Then we need to replace it with a "hidden field":

<%= f.hidden_field :article_id %>

With that done, you can now create comments by clicking a button on the article you want to comment on. When you then go to the index on comments, you should see the correct article_id associated with each comment.

That's all for now!