Creating Tags

Tags in a ticket-tracking application are extremely useful for making similar tickets easier to find and manage. In this article, we will create the interface for adding tags to a new ticket. This involves adding a new field to the new ticket page and defining a has_and_belongs_to_many association between the Ticket model and the not-yet-existent Tag model.

Creating tags feature

For this feature, we’re going to add a text field beneath the description field on the new ticket page, just like we see in figure 1.

![The tag box](image)

The words we enter into this field will become the tags for this ticket and we should see them on the ticket page. At the bottom of features/creating_tickets.feature, we’ll add a scenario that creates a new ticket with tags, shown in listing 1.

Listing 1 features/creating_tickets.feature

```ruby
Scenario: Creating a ticket with tags
  When I fill in "Title" with "Non-standards compliance"
  And I fill in "Description" with "My pages are ugly!"
  And I fill in "Tags" with "browser visual"
  And I press "Create Ticket"
  Then I should see "Ticket has been created."
  And I should see "browser" in "#ticket #tags"
  And I should see "visual" in "#ticket #tags"

When we run this scenario using bundle exec cucumber, it will fail features/creatingTickets.feature:48 declaring that it can't find the "Tags" field. Good! It's not there yet.
  And I fill in "Tags" with "browser visual"```

For source code, sample chapters, the Online Author Forum, and other resources, go to [http://www.manning.com/katz/](http://www.manning.com/katz/)
We’re going to need to take the data from this field and process each word into a new Tag object and, for that reason, we’ll use a text_field_tag to render this field. text_field_tag is similar to a text_field does; instead it will just output an input_tag with the type attribute set to "text" and the name set to whatever name we give it.

Using text_field_tag

To define this field, we will put the following code underneath the p tag for the description in app/views/tickets/_form.html.erb.

```erb
<p>
  <%= f.label_tag :tags %>
  <%= f.text_field_tag :tags, params[:tags] %>
</p>
```

This field will be sent through to TicketsController as simply params[:tags]. By specifying params[:tags] as the second argument to this method, we can repopulate this field when the ticket cannot be created due to it failing validation.

When we re-run this scenario, it no longer complains about the missing "Tags" field, but now that it can't find the tags area for our ticket:

```
And I should see “browser” within “#{ticket.downcase} tags”
scope ‘/[^\d - “ticket”]/’/’[^\d - “tags”]’ not found ...
```

We will need to define a #tags element inside the #ticket element so that this part of the scenario will pass. This element will contain the tags for our ticket, which our scenario will assert as actually visible.

Showing tags

We can add this new element to app/views/tickets/show.html.erb by adding this simple line underneath where we render the ticket's description:

```
<% render @ticket.tags %>
```

This creates the #ticket #tags element our feature is looking for and will render the soon-to-be-created app/views/tags/_tag.html.erb partial for every element in the also-soon-to-be-created tags association on the @ticket object.

So what out of these two steps is our next one? We can run our scenario again to see that it cannot find the tags method for a Ticket object:

```
undefined method `tags` for #<Ticket:0x0..>
```

This method is the tags method, which we’ll be defining with a has_and_belongs_to_many association between Ticket objects and Tag objects. It will be responsible for returning a collection of all the tags associated with the given ticket, much like a has_many would. It works in the opposite direction also: allowing us to find out what tickets have a specific tag.

Defining the tags association

We can define the association has_and_belongs_to_many on the Ticket model by using this line, placed after a new line after the has_many definitions inside our Ticket model:

```ruby
has_and_belongs_to_many :tags
```

This association will rely on a join table that doesn’t yet exist, called tags_tickets. This table contains only two fields, which are both foreign keys for tags and tickets. By using a join table, many tickets can have many tags, and vice versa.

When we rerun our scenario we’re told that there’s no constant called tag yet:

```
uninitialized constant Ticket::Tag (ActionView::Template::Error)
```

In other words, there is no Tag model yet. We should define this now if we want to go any further.

The Tag model

Our Tag model will have a single field called name which should be unique. To generate this model and its related migration we will run the rails command like this:

```
rails g model tag name:string --timestamps false
```
The **timestamps** option passed here determines whether or not the model's migration is generated with timestamps. Because we've passed the value of `false` to this, there will be no timestamps added.

Before we run this migration, we will need to add the join table called `tags_tickets` to our database, which has two fields: one called `ticket_id` and the other `tag_id`. The table name is the pluralized names of the two models it is joining, sorted in alphabetical order. This table will have no primary key as we only need it to join the `tags` and `tickets` table. We are never going to look for individual records from this table.

To define the table we will put this `tags_tickets` in the `self.up` section of our `db/migrate/[timestamp]_create_tags.rb` migration:
```ruby
create_table :tags_tickets, :id => false do |t|
  t.integer :tag_id, :ticket_id
end
```

The `:id => false` option passed to `create_table` here tells Active Record to create the table without the id field.

We should also add `drop_table :tag_tickets` to the `self.down` method in this migration too so that, if we need to, we can undo this migration. Next, we will run the migration on our development database by running `rake db:migrate` and our test database by running `rake db:test:prepare`. This will create the `tags` and `tags_tickets` tables. When we run this scenario again with `bundle exec cucumber features/creating_tickets:48`, it is now satisfied that the `tags` method is defined and has now moved on to whining that it can’t find the tag we specified:

```
And I should see “browser” within “#ticket #tags”
Failed assertion, no message given. (MiniTest::Assertion)
```

This is because we’re not doing anything to associate the text from the “Tags” field to the ticket we’ve just created. We need to parse the content from this field into new `Tag` objects and then associate them with the ticket we are creating, which we’ll look at how to do right now.

### Displaying a ticket’s tags

The `params[:tags]` in `TicketsController`’s `create` is the value from our “Tags” field on `app/views/tickets/_form.html.erb`. This is also the field we need to parse into `Tag` objects and associate them with the `Ticket` object we are creating.

To do this, we will alter the `create` action by adding this line directly after `@ticket.save`:
```ruby
@ticket.tag!(params[:tags])
```

This method will parse the tags from `params[:tags]`, convert them into new `Tag` objects, and associate them with the ticket. We can define this new method at the bottom of our `Ticket` model like this:
```ruby
def tag!(tags)
  tags = tags.split(" ").map do |tag|
    Tag.find_or_create_by_name(tag)
  end
  self.tags << tags
end
```

On the first line here, we use the `split` method to split our string into an array, and then the `map` method to iterate through every value in the array. Inside the block for `map`, we use a *dynamic finder* to find or create a tag with a specified name. `find_or_create_by_name` methods will always return a record, whether it be a preexisting or a recently created one.

After all the tags have been iterated through, we assign them to a ticket by using the `<<` method on the `tags` association. The `tag!` method we have just written will create the tags that we will display on the `app/views/tickets/show.html.erb` view by using this line:
```erb
<%= render @ticket.tags %>
```

When we run this scenario again by running `bundle exec cucumber features/creating_tickets.feature:48`, we will see it’s this line that is failing with an error:

```
Missing partial Tags/tag ...
```

Therefore, the next step is to write the tag partial, which our feature has just complained about. To do this, we will put the following code in `app/views/tags/_tag.html.erb`:
```erb
<span class="tag"><%= tag.name %></span>
```
By wrapping the tag name in a span with the of tag, class, it will be styled as defined in our stylesheet. With this partial defined, this puts the final piece of the puzzle for this feature into place. When we run our scenario again by running `bundle exec cucumber features/creating_tickets.feature:48` it passes:

1 scenario (1 passed)  
15 steps (15 passed)  

Great! This scenario is now complete. When a user creates a ticket, they are able to assign tags to that ticket and they display along with the ticket’s information on the show action for TicketsController.

![Test](image)

Figure 1 Look ma, a tag!

We will now commit this change, but, of course, before we do, we’ll ensure that we haven’t broken anything by running `rake cucumber:ok spec`.

53 scenarios (53 passed)  
620 steps (620 passed)  
# and  
27 examples, 0 failures, 10 pending

Good to see that nothing’s blown up this time. Let’s commit this change.

```
git add .  
git commit -m "Users can tag tickets upon creation"  
git push
```

Now that users are able to add a tag to a ticket when that ticket’s being created, we should also let them add tags to a ticket when they create a comment as well.

When a ticket’s discussion happens, new information may come about that would require that another tag be added to the ticket to group it into a different set. A perfect way to let our users do this would be to let them add it when they comment.

**Summary**

We’ve covered how to use a has_and_belongs_to_many association to define a link between tickets and tags. Tickets are capable of having more than one tag, but a tag is also capable of having more than one ticket assigned to it and, therefore, we use this type of association. A has_and_belongs_to_many could be used to associate people and the locations they’ve been.
Here are some other Manning titles you might be interested in:

- **Grails in Action**
  Glen Smith and Peter Ledbrook

- **Griffon in Action**
  Andres Almiray and Danno Ferrin

- **Spring in Action, Third Edition**
  Craig Walls

Last updated: March 11, 2011