Episode 234

Simple Form

A few months ago a couple of episodes covered the Formtastic gem [watch[1], read[2]]. This offers a convenient way to generate the view code for forms in Rails. In this episode we'll take a look at an alternative gem called SimpleForm[3]. Like Formtastic SimpleForm provides a simpler way to generate view code compared to standard Rails view code. If you know Formtastic then SimpleForm's method calls will seem familiar as they are very similar.

The big question here is given that the gems are so similar why would you choose SimpleForm over Formtastic? One reason is that SimpleForm is more lightweight. It doesn't include a stylesheet that it expects you to use instead just concentrating on generating HTML markup. It is also more customizable and extendable than Formtastic. If you've used Formtastic and feel that it gets in your way a little and forces you to use markup that you'd rather not then it's well work taking a look at SimpleForm.

Integrating SimpleForm Into an Application

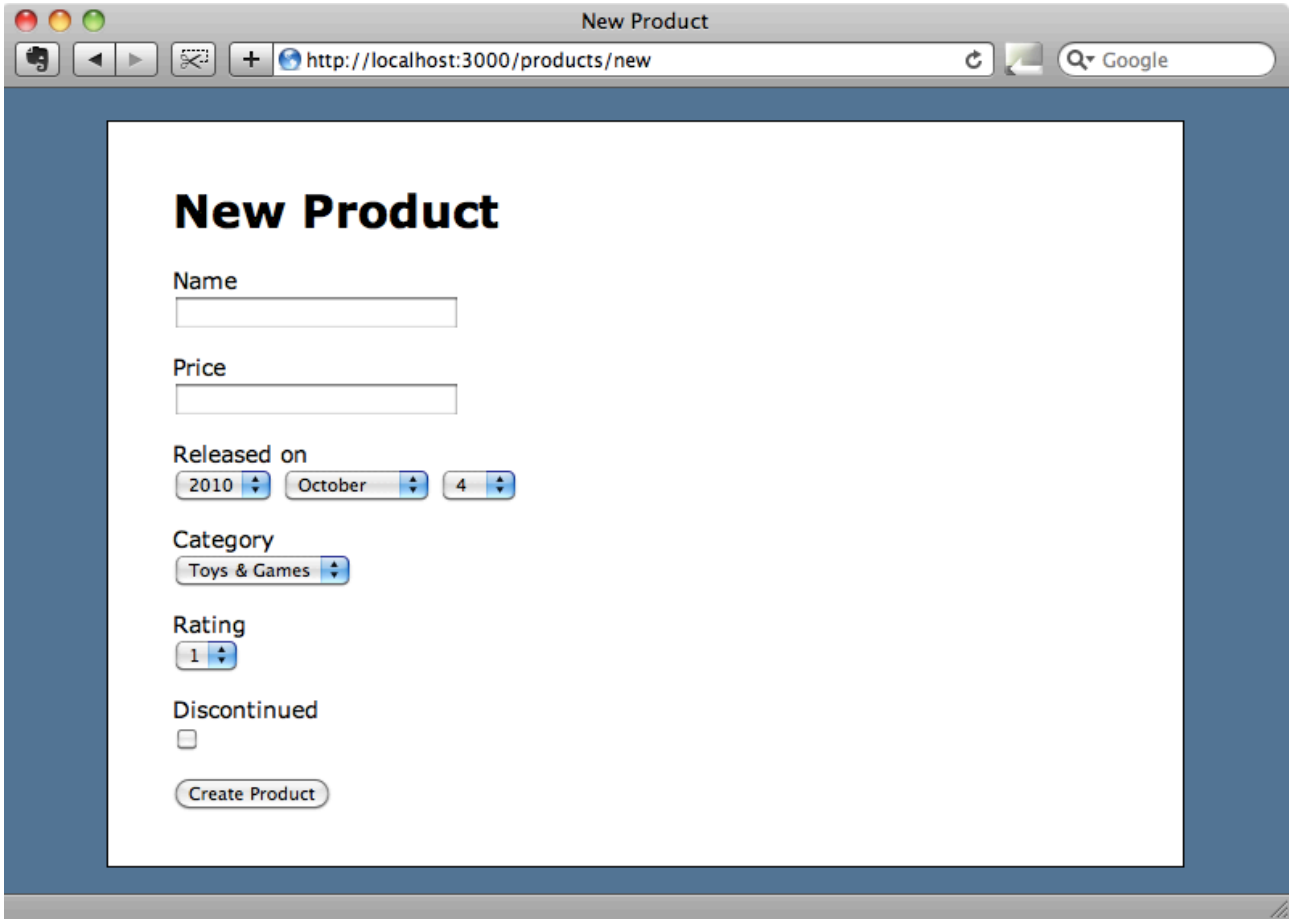
To demonstrate SimpleForm we'll update a form written using standard Rails view code to use SimpleForm instead. The application is a simple e-commerce application with a number of products, each of which belongs to a category and the form we'll be modifying is the form for creating a new product. This form has a number of different types of field so will provide a good example to convert to use with SimpleForm.

---

[1] http://railscasts.com/episodes/184-formtastic-part-1

[2] http://asciicasts.com/episodes/184-formtastic-part-1

[3] http://github.com/plataformatec/simple_form



To use SimpleForm we must first add the gem to the application's `Gemfile`.

`/Gemfile`

```
gem "simple_form"
```

Then, to ensure that it is installed, we need to run

```
$ bundle install
```

Now that the gem is installed we need to run a generator that will add a few files that SimpleForm needs into our application.

```
$ rails g simple_form:install
    create  config/initializers/simple_form.rb
    create  config/locales/simple_form.en.yml
    create  lib/templates/erb/scaffold/_form.html.erb
```

These generated files are used mainly for customization and we'll take a look at them shortly. Before we do that we'll update our product's form view code which currently looks like this.

/app/view/products/new.html.erb

```
<%= form_for @product do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :name %>
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :price %>
    <%= f.text_field :price %>
  </p>
  <p>
    <%= f.label :released_on %>
    <%= f.date_select :released_on %>
  </p>
  <p>
    <%= f.label :category_id %>
    <%= f.collection_select :category_id, Category.all, :id, :name
%>
  </p>
  <p>
    <%= f.label :rating %>
    <%= f.select :rating, 1..5 %>
  </p>
  <p>
    <%= f.label :discontinued %>
    <%= f.check_box :discontinued %>
  </p>
  <p><%= f.submit %></p>
<% end %>
```

The two main changes we need to make to the form are to replace `form_for` with `simple_form_for` and to replace each pair of labels and inputs with one of SimpleForm's methods.

/app/view/products/new.html.erb

```
<%= simple_form_for @product do |f| %>
  <%= f.error_messages %>
  <%= f.input :name %>
  <%= f.input :price %>
  <%= f.input :released_on %>
  <%= f.association :category %>
  <%= f.input :rating %>
  <%= f.input :discontinued %>
  <%= f.button :submit %>
<% end %>
```

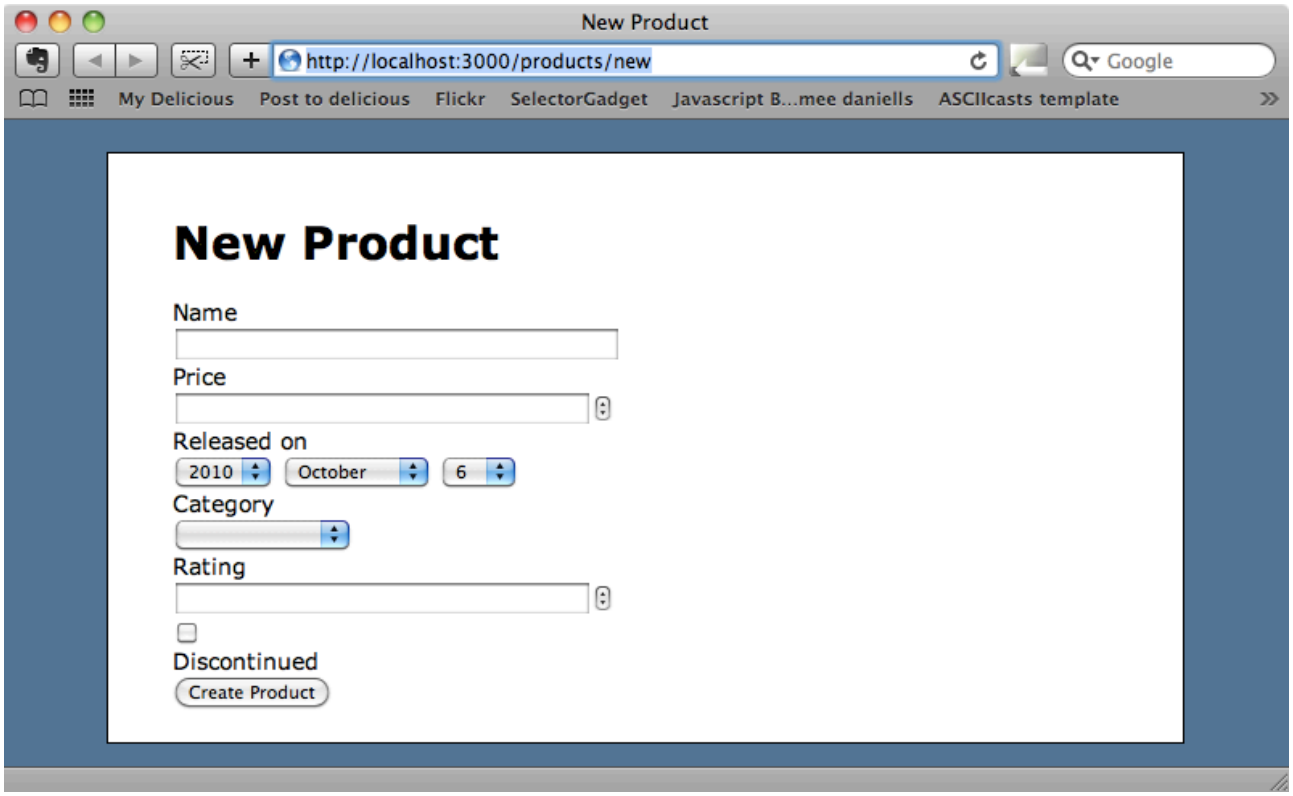
For most of the controls on the form we can use the `f.input` method. One field where we need to use something different is `category`. In the original form this field used `collection_select` to render a drop down menu containing all of the categories.

/app/view/products/new.html.erb

```
<%= f.collection_select :category_id, Category.all, :id, :name %>
```

With SimpleForm we can replace this with `f.association` which will do the same thing.

We can now reload the form and see how it is rendered by SimpleForm.



The form looks quite a bit different now as it no longer has the styling that was previously applied to it but it will be easy enough to alter our stylesheet to add some styles that match the class names that SimpleForm uses in its forms.

/public/stylesheets/application.css

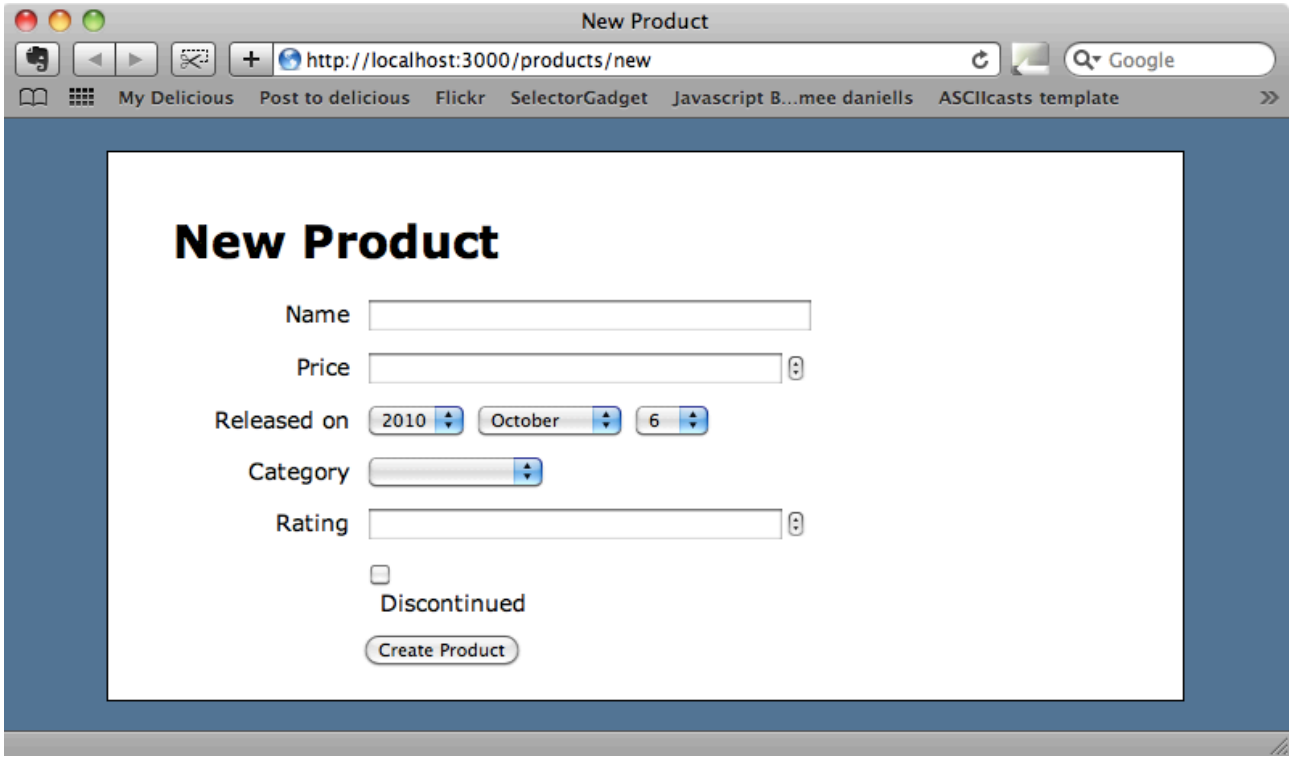
```
.simple_form label {
  float: left;
  width: 100px;
  text-align: right;
  margin: 2px 10px;
}

.simple_form div.input {
  margin-bottom: 10px;
}

.simple_form div.boolean, .simple_form input[type='submit'] {
  margin-left: 120px;
}

.simple_form div.boolean label {
  float: none;
  margin: 0;
}
```

When you use SimpleForm in your own application's you'll want to customize these styles to suit the look of your own application but the CSS above will give you some idea as to what class names are used in the HTML elements that SimpleForm generates. With the styles above in place our form now looks a little better.



Customizing Fields

For the fields that we have used `f.input` on SimpleForm has automatically detected each type of column so that "Released on" is rendered as a date column and "Discontinued" is rendered as a checkbox as it represents a boolean column. As an association the "Category" field has been rendered as a drop down menu with a blank option. If we want to customize these fields then we can do so by adding some options. For example we can remove the blank option from "Category" by adding an `:include_blank` option.

/app/view/products/new.html.erb

```
<%= f.association :category, :include_blank => false %>
```

The original form had a drop down list for the rating field, but this has now been replaced by a text field. We can restore this back to a drop down by using the `:collection` option and passing it a range.

/app/view/products/new.html.erb

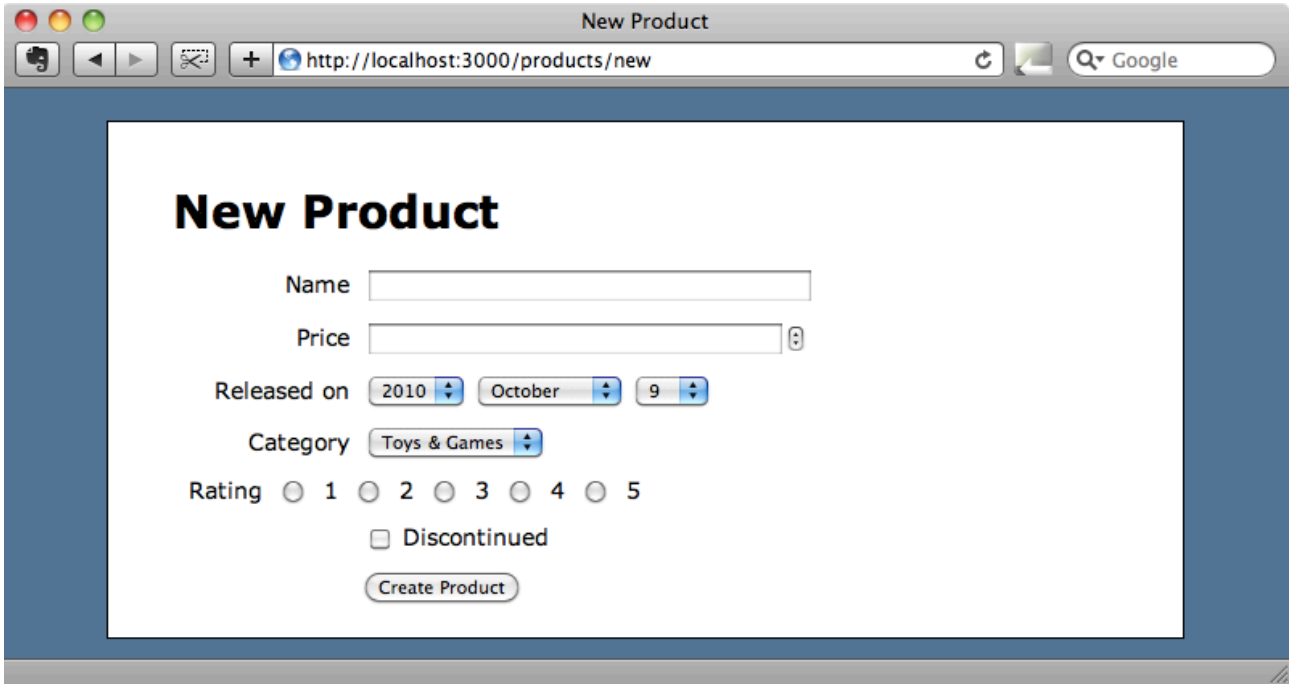
```
<%= f.input :rating, :collection => 1..5 %>
```

If we want the list rendered as radio buttons instead we can use the `:as` option.

/app/view/products/new.html.erb

```
<%= f.input :rating, :collection => 1..5, :as => :radio %>
```

When we reload the form now we'll see that the blank option has gone from the drop down and that the rating fields are rendered as radio buttons.

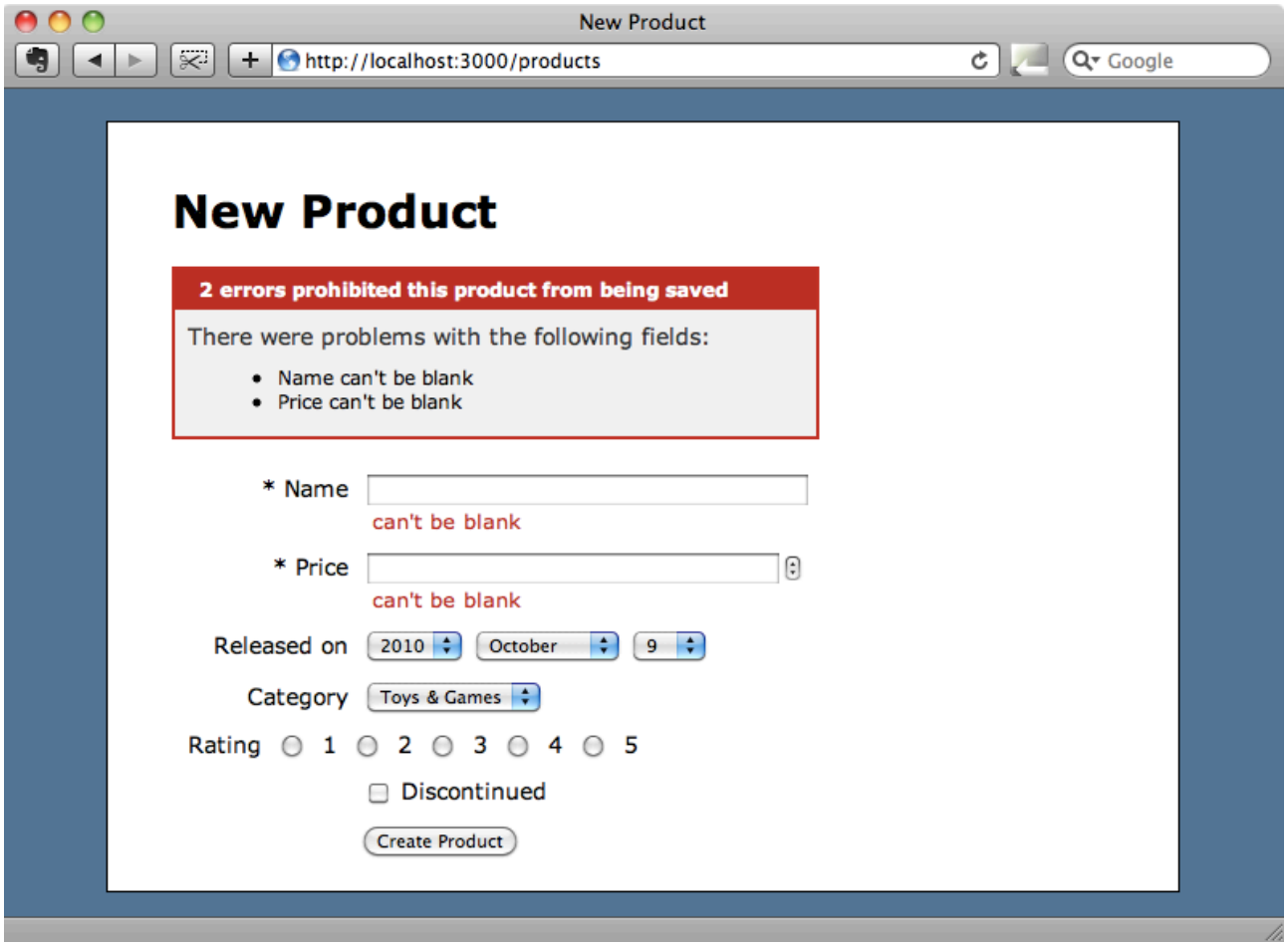


Another great feature of SimpleForm is automatic detection of required fields. We can demonstrate this by making the `name` and `price` fields in the `Product` model required.

/app/models/product.rb

```
class Product < ActiveRecord::Base
  attr_accessible :name, :price, :released_on, :category_id,
  :rating, :discontinued
  belongs_to :category
  validates_presence_of :name, :price
end
```

That's all we need to do. when we reload the form we'll see those fields marked with asterisks and if we try to submit it without filling in those fields we'll see the error messages for those fields displayed.

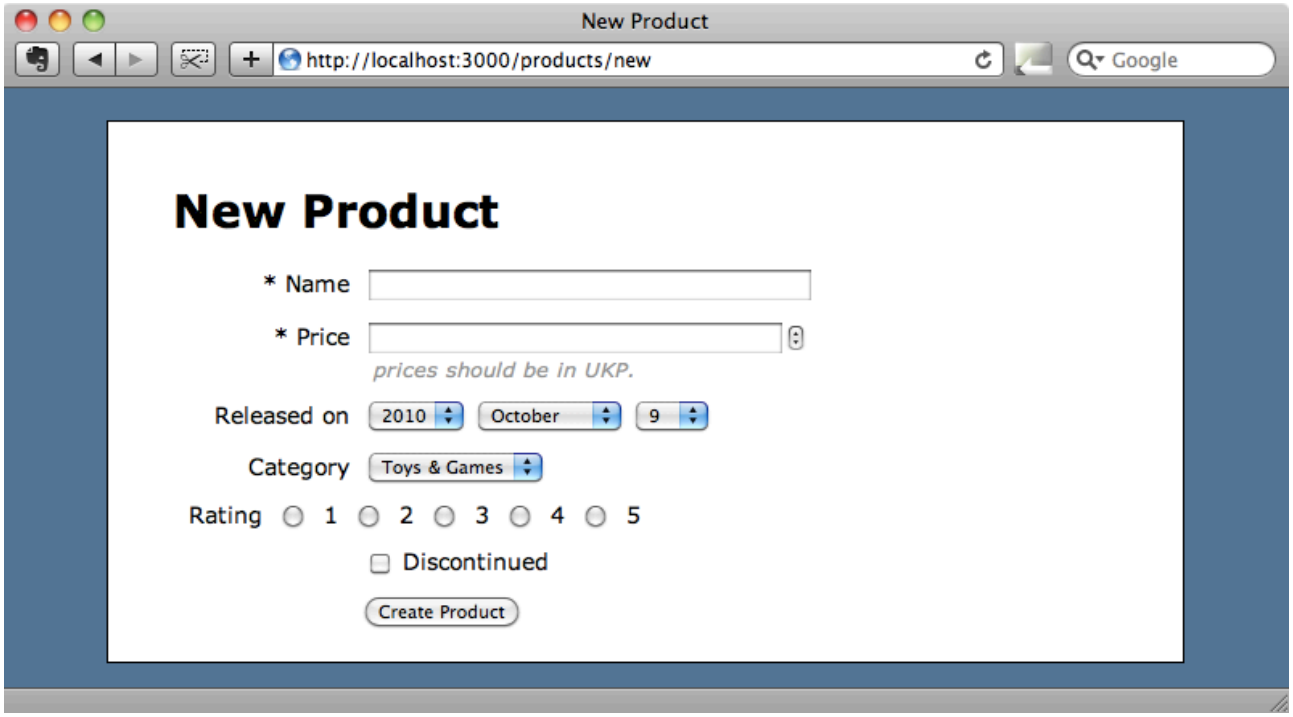


We can also easily add a hint message to each field. Let's say that we want to give a hint that prices should be entered in UK pounds. We just need to add a `:hint` option to the field.

/apps/views/products/_form.html.erb

```
<%= f.input :price, :hint => "prices should be in UKP." %>
```

This will add the hint after the form field.



There is more information on the options that can be passed to these fields in the README documentation[4] for SimpleForm. It's worth looking at the documentation for Formtastic[5] too as many of the options that it uses can be used by SimpleForm too.

Customizing SimpleForm

When we ran the SimpleForm generator earlier it created three files.

```
$ rails g simple_form:install
    create  config/initializers/simple_form.rb
    create  config/locales/simple_form.en.yml
    create  lib/templates/erb/scaffold/_form.html.erb
```

If we want to customize SimpleForm across an entire application then we can do so by modifying one or more of these files. The first file contains a number of commented-out configuration options and there are a lot of configuration changes we can make here. For example we can alter the order that each fields components are rendered; we can change that tag that wraps each form field and we can change the default size of the text field. We want narrower text fields in our application so we'll change the default value of `50` to `30`.

---

[4] http://github.com/plataformatec/simple_form#readme

[5] http://github.com/justinfrench/formtastic#readme

/config/initializers/simple_form.rb

```
# Use this setup block to configure all options available in
SimpleForm.
SimpleForm.setup do |config|

  # Other options removed...

  # Default size for text inputs.
  config.default_input_size = 30
end
```

The next file that is generated is the locales file and this is where we can internationalize the form.

/config/locales/simple_form.en.yml

```
en:
  simple_form:
    "yes": 'Yes'
    "no": 'No'
    required:
      text: 'required'
      mark: '*'
      # You can uncomment the line below if you need to overwrite
the whole required html.
      # When using html, text and mark won't be used.
      # html: '<abbr title="required">*</abbr>'
    error_notification:
      default_message: "Some errors were found, please take a
look:"
    # Labels and hints examples
    # labels:
    #   password: 'Password'
    #   user:
    #     new:
    #       email: 'E-mail para efetuar o sign in.'
    #     edit:
    #       email: 'E-mail.'
    # hints:
    #   username: 'User name to sign in.'
    #   password: 'No special characters, please.'
```

Even if your application doesn't need to display multiple languages this file is still a useful place to alter some of SimpleForm's options such as the text that is displayed against required fields.

The final file that is generated overrides the form partial for the scaffold generator.

/lib/templates/erb/scaffold/_form.html.erb

```
<%%= simple_form_for(@<%= singular_name %>) do |f| %>
  <%% if @<%= singular_name %>.errors.any? %>
    <div id="error_explanation">
      <h2><%%= pluralize(@<%= singular_name %>.errors.count,
"error") %> prohibited this <%= singular_name %> from being
saved:</h2>

      <ul>
      <%% @<%= singular_name %>.errors.full_messages.each do |msg|
%>
        <li><%%= msg %></li>
      <%% end %>
      </ul>
    </div>
  <%% end %>

  <div class="inputs">
  <%- attributes.each do |attribute| -%>
    <%%= f.<%= attribute.reference? ? :association : :input %> :<
%= attribute.name %> %>
  <%- end -%>
  </div>

  <div class="actions">
    <%%= f.button :submit %>
  </div>
<%% end %>
```

In Rails 3 we can override any template file in a generator by adding it to our application so if we want to customize the way a generator works to fit the way we work we can just create a new template file.

That's it for this episode. SimpleForm makes it easy to create forms in your Rails applications and is well worth taking a look at.