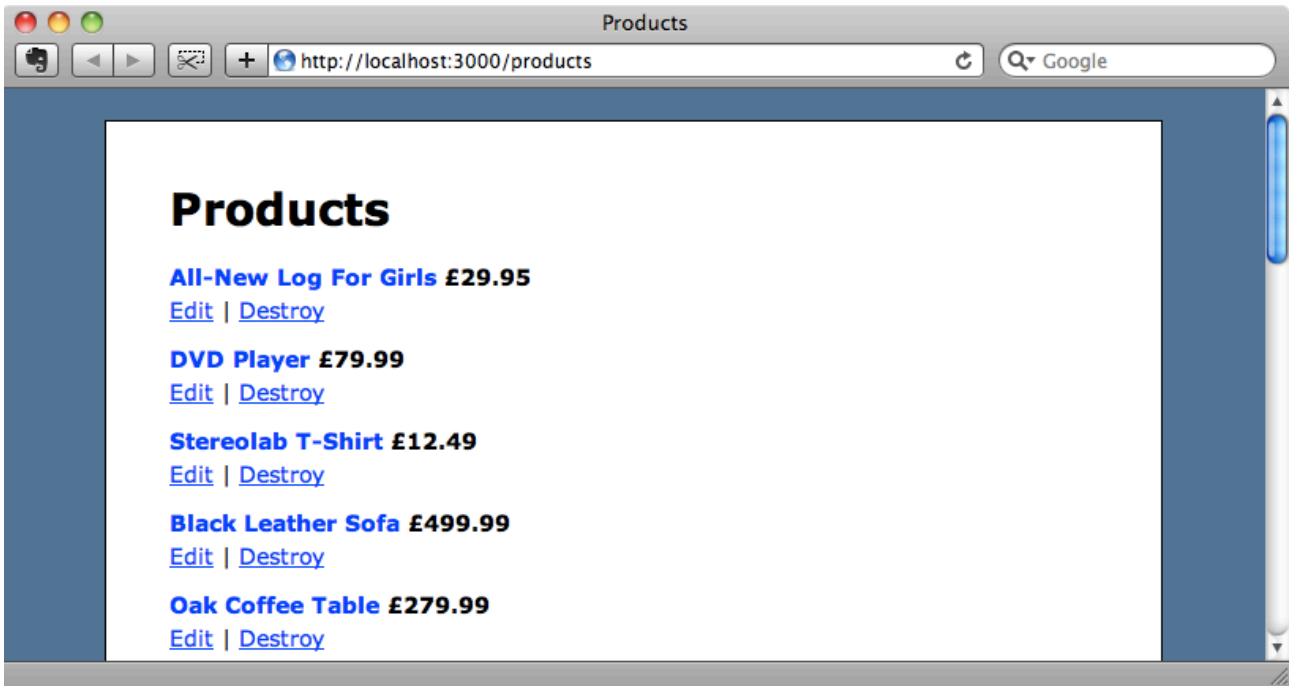
Episode 254

Pagination with Kaminari

Below is a screenshot from a Rails 3 application that shows a long list of products. Rather than show the items as a single list we'd like to use pagination.



The first-choice gem for pagination in Rails is will_paginate[1], but the currently released version doesn't support Rails 3. There is a pre-release version available that works but it hasn't been updated for several months. If will_paginate is no longer in active development are there any other gems we could use?

One alternative is Kaminari[2]. This seems to provide a cleaner implementation of pagination and offers several improved features, too, so let's try it in our application instead. Kaminari is installed in the usual way: first by adding a reference to it in the application's `Gemfile` and then running `bundle` to make sure that the gem is installed on our system.

/Gemfile

```
gem "kaminari"
```

[1] https://github.com/mislav/will_paginate/wiki

[2] https://github.com/amatsuda/kaminari

Using Kaminari

Kaminari provides a scope called `page` that can be applied to any ActiveRecord model. We can use it in the `ProductsController`'s `index` action to page the list of products, passing in the page we want to view. We'll pass in the `page` parameter from the query string.

/app/controllers/products_controller.rb

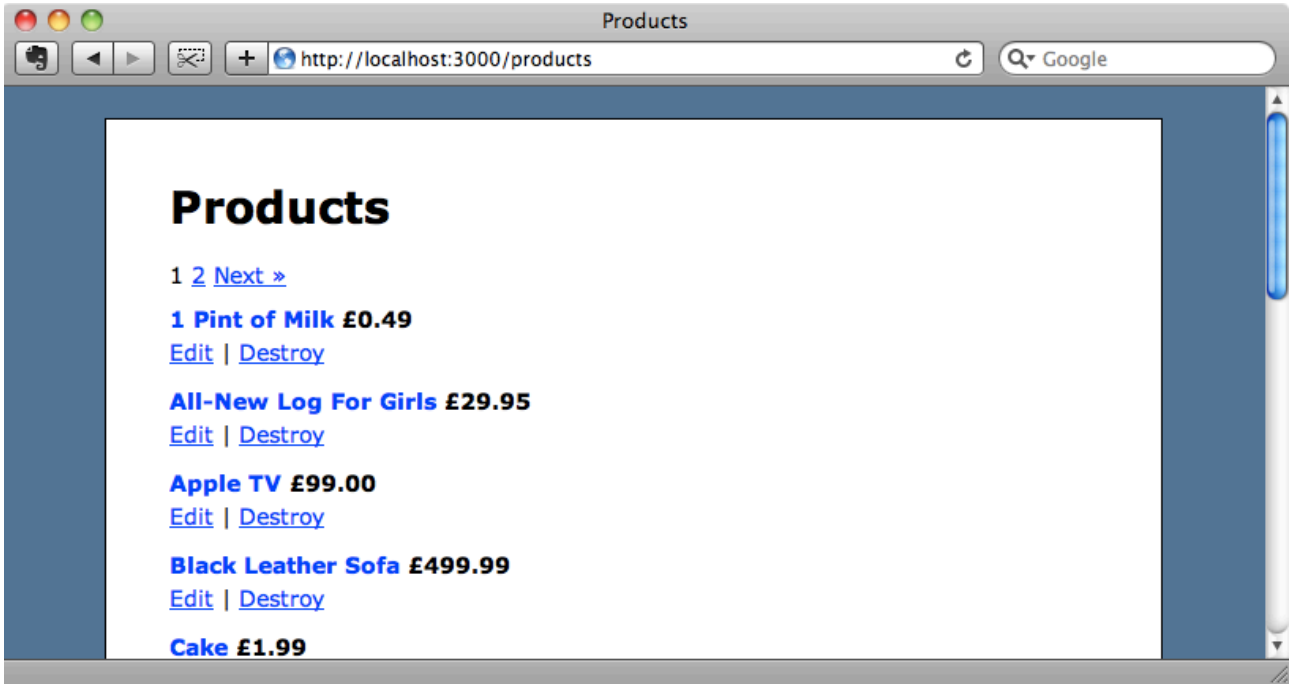
```
def index
    @products = Product.order("name").page(params[:page])
end
```

Inside the `index` template where we're rendering the list of products we can call the `paginate` helper method that Kaminari provides, passing in the list we're paginating.

/app/views/products/index.html.erb

```
<% title "Products" %>
<%= paginate @products %>
<div id="products">
  <%= render @products %>
</div>
<p><%= link_to "New Product", new_product_path %></p>
```

If we reload the page now we'll see the pagination links at the top of the page.

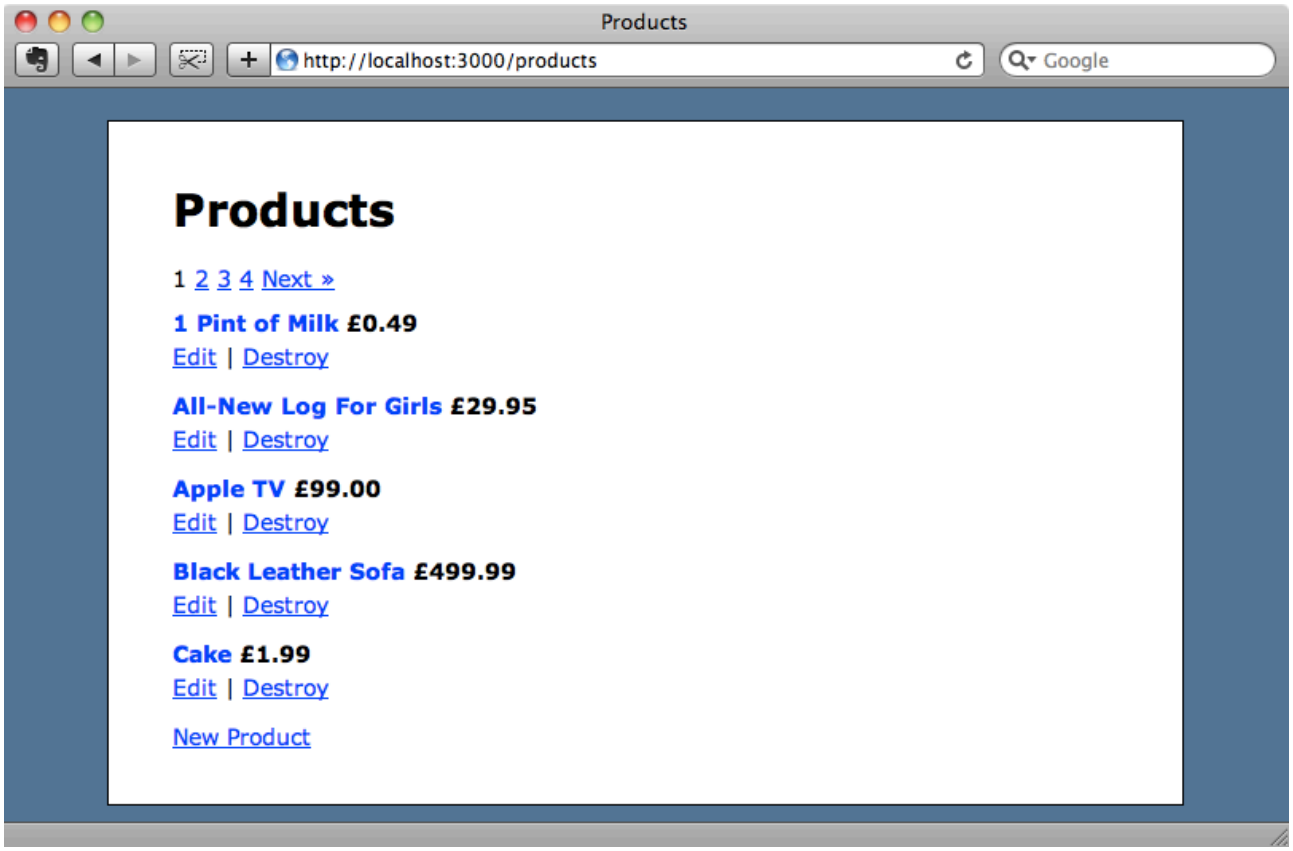


Kaminari will show 25 items per page by default but we can easily change that by calling another scope called `per`. We'll use this now to change the number of items per page to 5.

/app/controllers/products_controller.rb

```
def index
  @products = Product.order("name").page(params[:page]).per(5)
end
```

Now we have five items per page.



This syntax flows nicely with the new ActiveRecord queries in Rails 3. We could move the `order` call anywhere within the query and it would work just the same as all we're doing is chaining scopes against our model. We could even move the `page` and `per` calls into other named scopes in the model and it would work as expected. One exception to this is that `per` needs to be called after `page`. The reason for this is that `page` adds the `per` scope. If we call `per` before `page` we'll get an undefined method error as it won't have been added yet.

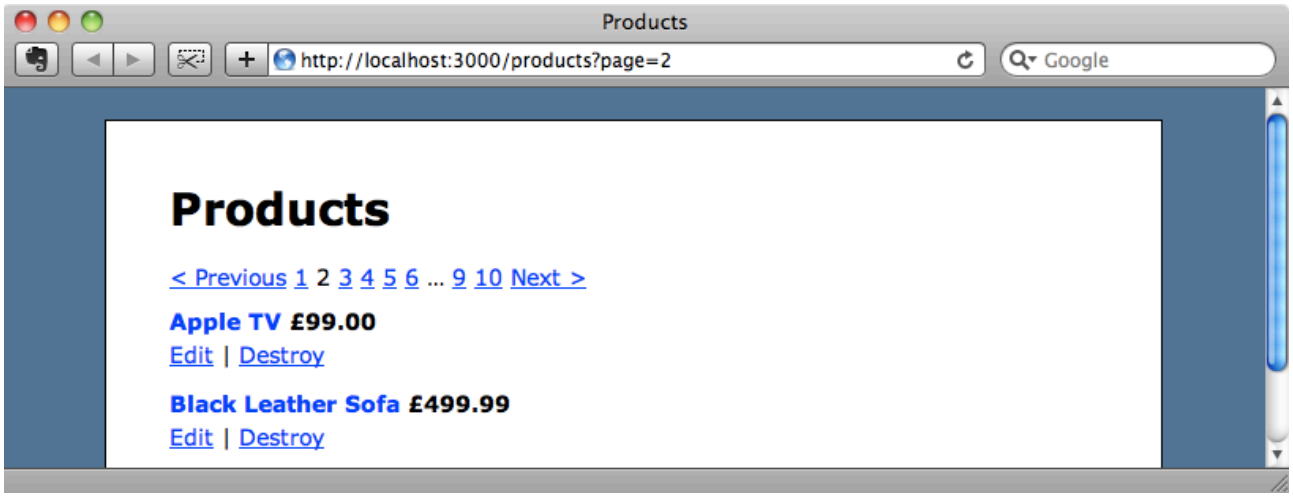
Changing The Look

There are several ways to change the way that the pagination looks. We want to change the words "prev" and "next" in the list of pages and the arrows that go with them and we can do so by adding entries into the localization file. This makes it easy to alter these items, especially if your application needs to support multiple languages. The items we need to add go under `views/pagination` and we can change the previous and next text by setting the `previous` and `next` keys. To change the text that displays when the list of pages is truncated we set the `truncate` key. Note that we have to escape the HTML here as it isn't escaped when shown on the page.

/config/locales/en.yml

```
en:
  hello: "Hello world"
  views:
    pagination:
      previous: "&lt; Previous"
      next: "Next &gt;"
      truncate: "&hellip;”
```

When we reload the page we'll see the changed text.



We can make additional changes by adding some CSS for the classes that the different parts of the navigation use. One thing worth noting is that the navigation uses the HTML 5 nav element.

```
<nav class='pagination'>
  <span class="prev">
    <a href="/products" class="prev" rel="prev">&lt; Previous</a>
  </span>
  <span class="page first">
    <a href="/products">1</a>
  </span>
  <span class="page current">2</span>
  ...
```

While we can change the appearance of the navigation quite a bit with CSS there are some changes that can't be done this way. For example we might want the "Previous" link to be visible on the first page, but greyed-out and disabled, rather than being hidden as it is by default.

Kaminari is a Rails Engine and it comes with a number of view files which can be customized to suit our application. To help with this we can use a generator provided by Kaminari. We need to pass in the name of a theme to this generator as an argument. If we use `default` it will use the theme that Kaminari comes with.

```
$ rails g kaminari:views default
      create  app/views/kaminari/_current_page.html.erb
      create  app/views/kaminari/_first_page_link.html.erb
      create  app/views/kaminari/_last_page_link.html.erb
      create  app/views/kaminari/_next_link.html.erb
      create  app/views/kaminari/_next_span.html.erb
      create  app/views/kaminari/_page_link.html.erb
      create  app/views/kaminari/_paginator.html.erb
      create  app/views/kaminari/_prev_link.html.erb
      create  app/views/kaminari/_prev_span.html.erb
      create  app/views/kaminari/_truncated_span.html.erb
```

There are other themes listed at the Kaminari Themes[3] project on GitHub. Only a couple are available at the time of writing but more will no doubt appear soon as there is enough there to get started and to develop your own themes.

The generator creates a new `kaminari` directory under `app/views` and puts a number of partial files there. We can modify these files to customize the behaviour of the pagination. The main file is `_paginator.html.erb` and while it looks complicated at first, it's fairly easy to understand and modify.

[3] https://github.com/amatsuda/kaminari_themes

/app/views/kaminari/_paginator.html.erb

```
<%# The container tag
  - available local variables
    current_page:  the page number of currently displayed page
    num_pages:     total number of pages
    per_page:      number of items to fetch per page
    remote:        data-remote
    paginator:     the paginator that renders the pagination tags
inside
-%>
<%= paginator.render do -%>
  <nav class='pagination'>
    <%= current_page > 1 ? prev_link_tag : prev_span_tag %>
    <% each_page do |page| -%>
      <% if page.current? -%>
        <%= current_page_tag %>
      <% elsif page.left_outer? || page.right_outer? ||
page.inside_window? -%>
        <% if page.first? -%>
          <%= first_page_link_tag %>
        <% elsif page.last? -%>
          <%= last_page_link_tag %>
        <% else -%>
          <%= page_link_tag %>
        <% end -%>
      <% elsif !page.was_truncated? -%>
        <%= truncated_span_tag %>
      <% end -%>
    <% end -%>
    <%=num_pages > current_page ? next_link_tag : next_span_tag %>
  </nav>
<% end -%>
```

It's easy to see the `nav` element on the page and the code immediately below it that renders the "previous" link. This code will call either `prev_link_tag` or `prev_span_tag` depending on whether the current page is the first page. Each of these methods calls a partial file. We want to change what happens on the first page so we'll take a look at `_prev_span.html.erb`.

/app/view/kaminari/_prev_span.html.erb

```
<%# "Previous" without link
  - available local variables
    current_page:  the page number of currently displayed page
    num_pages:     total number of pages
    per_page:      number of items to fetch per page
    remote:        data-remote
-%>
<span class="prev"></span>
```

All we need to do is add some code inside the `span` element. If we look in `_prev_link.html.erb` we'll see that it renders a link. We can copy the code that generates the text for that link and paste it into the span element.

/app/view/kaminari/_prev_span.html.erb

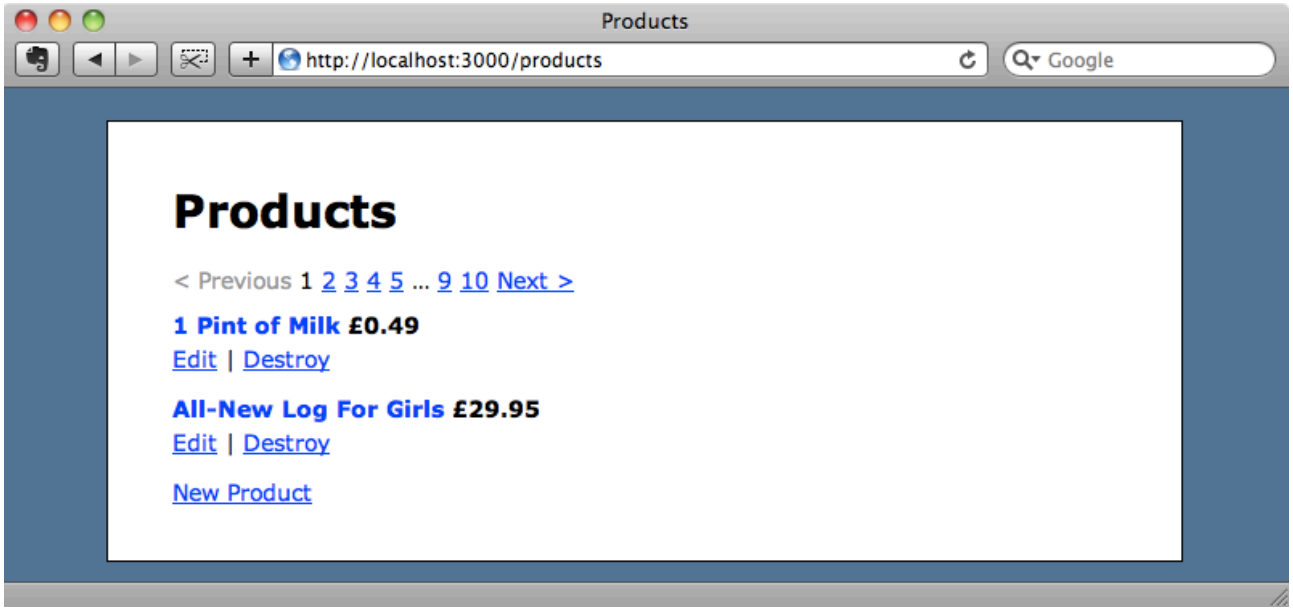
```
<%# "Previous" without link
  - available local variables
    current_page:  the page number of currently displayed page
    num_pages:     total number of pages
    per_page:      number of items to fetch per page
    remote:        data-remote
-%>
<span class="prev disabled">
  <%= raw(t 'views.pagination.previous') %>
</span>
```

The code that renders the text uses the `t` method to get a piece of localized text from the localizations that we modified earlier and that text is passed to `raw` so that it isn't escaped. We've also added a `disabled` class to the `span` so that we can show it greyed out by adding the following CSS.

/public/stylesheets/application.css

```
.disabled { color: #999; }
```

When we load the first page of products now we'll see the greyed-out text.



Obviously we'd want to do the same for the "Next" link on the last page of items but we won't show that here.

That's it for our look at the Kaminari gem. If you need pagination in a Rails 3 application it's well worth taking a look at. We haven't covered the various options that can be passed in to the `paginate` helper method, such as the ability to change the way that the list of pages is displayed, adding parameters to the pagination links' URLs or making the links work via AJAX.