

[Ruby \(http://www.sitepoint.com/ruby/\)](http://www.sitepoint.com/ruby/)

Breadcrumbs on Rails with Gretel



[\(http://www.sitepoint.com/author/ibodrov/\)](http://www.sitepoint.com/author/ibodrov/)

[Ilya Bodrov-Krukowski \(http://www.sitepoint.com/author/ibodrov/\)](http://www.sitepoint.com/author/ibodrov/)

Published August 25, 2014

[Tweet \(Https://twitter.com/share?text=Breadcrumbs+on+Rails+with+Gretel&via=sitepointdotcom\)](https://twitter.com/share?text=Breadcrumbs+on+Rails+with+Gretel&via=sitepointdotcom)

[Subscribe \(Https://confirmsubscription.com/h/y/1FD5B523FA48AA2B\)](https://confirmsubscription.com/h/y/1FD5B523FA48AA2B)



You probably know that breadcrumbs are navigational elements that help users track their current location on a website. Breadcrumbs can list either previously visited pages or parent pages of the current one (on hierarchical websites). You also probably know where this term comes, but I'll remind you just to be sure.

Wikipedia says that "Hansel and Gretel" is a well-known fairy tale of German origin recorded by Brothers Grimm and published in 1812. This tale tells about a young brother and sister who were left in the woods by their own father because the family had nothing to eat. Hansel, fortunately, happened to have a slice of bread with him and left a trail of bread crumbs so that they could find a way back home. Unfortunately, all the crumbs were eaten by birds and the children had to keep wandering, ending up facing a wicked witch. If you don't know how this tale ends, read it some time.

Anyway, the term “breadcrumbs” has its origin in this tale. Just like Hansel left a trail of crumbs, we present a user with a trail of hyperlinks so they can see the trail they’ve taken through the site. Let’s hope our breadcrumbs are not eaten by birds as well...

Having breadcrumbs on a large hierarchical websites (like the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web) (<https://developer.mozilla.org/en-US/docs/Web>)) is a nice practice, but how can we integrate it into our Rails app?

This article creates a demo app with breadcrumbs using a gem called Gretel. The demo shows how breadcrumbs can be styled, customized, and scaled for large websites.

The source code for the demo app is available on [GitHub](https://github.com/bodrovis/SitepointGretel) (<https://github.com/bodrovis/SitepointGretel>).

A working demo can be found at <http://sitepoint-gretel.herokuapp.com/> (<http://sitepoint-gretel.herokuapp.com/>).

Starting the Journey

Rails 4.1.4 is used for this demo, but the same solution can be implemented with Rails 3.

Okay, let’s go. We’ll create Music Shop, a really simple, online store selling music albums. Of course, we won’t implement everything, but let’s pretend that some day this app will grow into something big.

Create a new app without default testing suite:

```
1 | $ rails new music_shop -T
```

We’ll use [Twitter Bootstrap](http://getbootstrap.com/) (<http://getbootstrap.com/>) for style and layout, so drop this gem into your *Gemfile*:

Gemfile

```
1 | [...]
2 | gem 'bootstrap-sass'
```

Run

```
1 | $ bundle install
```

Next, rename `application.css` to `application.css.scss` and change its contents to:

stylesheets/application.css.scss

```
1 @import 'bootstrap';
2 @import 'bootstrap/theme';
```

To take advantage of Bootstrap's magic, change the layout:

layouts/application.html.erb

```
1 [...]
2 <div class="navbar navbar-inverse" role="navigation">
3   <div class="container">
4     <div class="navbar-header">
5       <%= link_to 'Music shop', root_path, class: 'navbar-brand' %>
6     </div>
7     <ul class="nav navbar-nav">
8       <li><%= link_to 'Albums', albums_path %></li>
9     </ul>
10  </div>
11 </div>
12
13 <div class="container">
14   <%= yield %>
15 </div>
16 [...]
```

Okay, now we can move on to the model and corresponding routes. For this demo, there is a single model – `Album` – with the following fields (not to mention default `id` , `created_at` , `updated_at`):

- `title` (`string`)
- `description` (`text`)
- `artist` (`text`)
- `price` (`decimal`)

Create the corresponding migration:

```
1 $ rails g model Album title:string description:text artist:string price:decimal
```

Open the migration file and alter this line:

xx_create_albums.rb

```
1 [...]
2 t.decimal :price, precision: 6, scale: 2
3 [...]
```

The price can have up to 6 digits with 2 digits after the decimal point. Apply the migration:

```
1 $ rake db:migrate
```

And some routes (we are going to skip `update` , `destroy` , `new` and `create` for now and pretend those will be created on the next iteration):

routes.rb

```
1 [...]
2 resources :albums, only: [:index, :edit, :show]
3 root to: 'albums#index'
4 [...]
```

Now, create the corresponding controller:

albums_controller.rb

```
1 class AlbumsController < ApplicationController
2   def index
3     @albums = Album.all
4   end
5
6   def edit
7     @album = Album.find(params[:id])
8   end
9
10  def show
11    @album = Album.find(params[:id])
12  end
13 end
```

Let's also take care of the `index` view:

albums/index.html.erb

```
1 <div class="page-header">
2   <h1>Albums</h1>
3 </div>
4
5 <table class="table table-hover">
6   <tr>
7     <th>Title</th>
8     <th>Artist</th>
9     <th>Description</th>
10    <th>Price</th>
11  </tr>
12  <% @albums.each do |album| %>
13    <tr>
14      <td><%= link_to album.title, album_path(album) %></td>
15      <td><%= album.artist %></td>
16      <td><%= album.description %></td>
17      <td><%= number_to_currency album.price, unit: '$' %></td>
18    </tr>
19  <% end %>
20 </table>
```

Note that we are using the `number_to_currency` helper to format the price correctly (we are presuming the price is entered in USD).

Adding Some Albums

At this point, we have the model, controller and a view, but no data. We could enter them by hand, but that would be really tedious. Instead, let's use `seeds.rb` to load some demo data into our database.

Generate some random titles and prices using the faker (<https://github.com/stympy/faker>) gem created by Benjamin Curtis. Add `faker` to the *Gemfile*:

Gemfile

```
1 [...]
2 gem 'faker'
3 [...]
```

and don't forget to run

```
1 $ bundle install
```

Create a script to load the demo data:

seeds.rb

```
1 50.times do
2    Album.create({title: Faker::Lorem.words(2).join(' ').titleize,
3                  description: Faker::Lorem.paragraph(3, true, 4),
4                  artist: Faker::Name.name, price: Faker::Commerce.price})
5 end
```

`words(2)` means that we generate an array of two random words. `titleize` is used to format those words like a title. `paragraph(3, true, 4)` creates three paragraphs with four random sentences.

`name`, as you've might have guessed, generates a random name and `price` creates a price. Easy, isn't it?

Load the data:

```
1 $ rake db:seed
```

Run the server and check out how this all looks. We've done the ground work and are ready to start integrating breadcrumbs.

Getting the Slice of Bread

Just like Hansel had his slice of bread, we need our own tool to start creating our bird-proof trail. Meet [Gretel \(https://github.com/lassebunk/gretel\)](https://github.com/lassebunk/gretel), the flexible Rails breadcrumbs plugin created by Lasse Bunk. As stated in this [nice summary image \(https://i.imgur.com/nH25yiH.png\)](https://i.imgur.com/nH25yiH.png), Gretel does not mess with your controller – all required configuration is placed elsewhere.

Drop the gem into your Gemfile:

Gemfile

```
1 [...]
2 gem 'gretel'
3 [...]
```

and run

```
1 $ bundle install
```

Use the generator to create the basic configuration file:

```
1 $ rails generate gretel:install
```

Inside the *config* directory, you will find the *breadcrumbs.rb* file that contains all your configuration. You are probably thinking, “This means that I will have to reload my server each time I modify this file?”. Fortunately, you don’t.

Starting from version 2.1.0, in the development environment this file gets reloaded automatically each time it is changed. In the production environment, it is loaded only once.

Now, we can create our first crumb using the following code:

breadcrumbs.rb

```
1 crumb :root do
2   link "Home", root_path
3 end
```

The layout and views also requires modifications:

layouts/application.html.erb

```
1 <%= breadcrumbs pretext: "You are here: " %>
```

The `pretext` option specifies the text that should be placed in front of the crumbs on the page. You can also specify `posttext` to put something after the crumbs.

albums/index.html.erb

```
1 <% breadcrumb :root %>
2 [...]
```

This line is required to tell which crumb to render.

Fire up the server, navigate to the main page, and... breadcrumbs will not show up. Why is that???

By default, Gretel does not show breadcrumbs if it has only one link. To fix this, alter the `breadcrumbs` method:

layouts/application.html.erb

```
1 | <%= breadcrumbs pretext: "You are here: ", display_single_fragment: true %>
```

You should now see your breadcrumbs block. Another problem is styling – right now it looks pretty ugly. Luckily, Gretel already knows how to render breadcrumbs for Bootstrap. Just provide the `style` option like this:

layouts/application.html.erb

```
1 | <%= breadcrumbs pretext: "You are here: ", display_single_fragment: true, style: :bootstrap %>
```

`style` may take other values like `ul`, `ol`, `foundation5` (renders breadcrumbs to be styled by Foundation (<http://foundation.zurb.com/>)) and `inline`.

Reload the page et voila! Bootstrap will take care of the styling for us. However, there is a problem with the separator between crumbs. This is because Bootstrap's styles append `/` as a separator and Gretel uses `>` by default. So, again, alter the `breadcrumbs` method:

layouts/application.html.erb

```
1 | <%= breadcrumbs pretext: "You are here: ", display_single_fragment: true, style: :bootstrap, separa
```

Cool! By the way, if you want the current crumb to be rendered as a link set the `link_current` option to `true`. You may also use semantic breadcrumbs (<https://support.google.com/webmasters/answer/185417?hl=en>) by setting `semantic` to `true`. Gretel has a handful of other options, which can be found here (<https://github.com/lassebunk/gretel#options>).

Creating the Trail

Okay, how about extending the trail a bit? We still have `show` and `edit` views to be created. Let's start with `show`:

albums/show.html.erb


```

1 <div class="page-header">
2   <h1><%= @album.title %></h1>
3 </div>
4
5 <p><%= @album.description %></p>
6
7 <%= link_to 'Edit', edit_album_path(@album), class: 'btn btn-primary' %>

```

In a real app, you'd need some kind of authentication and authorization system so that the “Edit” button is not visible to everyone, but this is just a demo.

Now create a new crumb:

breadcrumbs.rb

```

1 crumb :album do |album|
2   link album.title, album
3   parent :root
4 end

```

Here, the `album` variable is used to fetch the album's title and display it on the page. We also specify the crumb's parent as `root`. By the way, we could have skipped this line because, by default, Gretel makes `root` the parent of the crumb (it is controlled by the `autoroot` option).

Add the `breadcrumb` method to the 'show' view:

albums/show.html.erb

```

1 <% breadcrumb :album, @album %>
2 [...]

```

As you see here, we not only provide the crumb's name, but also pass the resource. Actually, we can provide as many arguments as we want. This line can also be simplified (breadcrumb will be automatically inferred):

```

1 <% breadcrumb @album %>

```

Lastly, let's deal with the `edit` view:

albums/edit.html.erb

```
1 | <% breadcrumb :edit_album, @album %>
2 |
3 | <div class="page-header">
4 |   <h1>Edit <%= @album.title %></h1>
5 | </div>
6 |
7 | <p>Coming in the next version...</p>
```

Don't forget to create a new crumb:

breadcrumbs.rb

```
1 | [...]
2 | crumb :edit_album do |album|
3 |   link "Edit #{album.title}", album
4 |   parent :album, album
5 | end
```

Again, we use the `Album` resource and specify the `album` crumb as a direct parent. Reload your page, navigate to an album, and click “Edit” – note how the breadcrumbs change. That was easy, huh?

Extending the Trail

Okay, let's create some more crumbs. As we've already discussed, the real online would have some kind of authentication. We are not going to do this, but we will create a user's profile page and add a crumb for it.

Create a new route:

routes.rb

```
1 | resources :users, only: [:show]
```

and the corresponding controller:

users_controller.rb

```
1 | class UsersController < ApplicationController
2 | end
```

Provide a link to the `show` page (with a fake user id):

layouts/application.html.erb

```
1 <div class="navbar navbar-inverse" role="navigation">
2   <div class="container">
3     <div class="navbar-header">
4       <%= link_to 'Music shop', root_path, class: 'navbar-brand' %>
5     </div>
6     <ul class="nav navbar-nav">
7       <li><%= link_to 'Albums', albums_path %></li>
8     </ul>
9
10    <ul class="nav navbar-nav navbar-right">
11      <li><%= link_to 'Profile', user_path(1) %></li>
12    </ul>
13    [...]
```

and create the actual view:

users/show.html.erb

```
1 <% breadcrumb :user %>
2 <div class="page-header">
3   <h1><%= current_user %>'s profile</h1>
4 </div>
5
6 <p>Coming soon...</p>
```

Suppose we have the `current_user` helper method that returns the user's name. Can we use this helper when creating the crumb? Let's try it out!

application_helper.rb

```
1 module ApplicationHelper
2   def current_user
3     "Someone"
4   end
5 end
```

breadcrumbs.rb

```
1 [...]
2 crumb :user do
3   link "#{current_user}'s profile"
4 end
```

It appears that helpers can be used without any problems! We can also implement conditions inside the `crumb` method. Suppose we want to include a link to the admin area in the breadcrumb if the user is an admin. Add this `admin?` stub method:

application_helper.rb

```
1 [...]
2 def admin?
3   true
4 end
5 [...]
```

And alter your crumb:

breadcrumbs.rb

```
1 [...]
2 crumb :admin_root do
3   link "Admin's area"
4   parent :root
5 end
6
7 crumb :user do
8   link "#{current_user}'s profile"
9   if admin?
10    parent :admin_root
11  else
12    parent :root
13  end
14 end
```

After reloading the page, you'll see the Admin area" crumb. Note that the second argument has not been provided to the `link` method, which causes the crumb to be rendered as plain text, without a link.

Okay, one more experiment. Imagine we have a search page and want to include breadcrumbs there as well. Could we show the actual search term inside the breadcrumb?

Add one more route:

routes.rb

```
1  [...]
2  get '/search', to: 'pages#search', as: :search
3  [...]
```

Create controller:

pages_controller.rb

```
1  class PagesController < ApplicationController
2    def search
3      @term = params[:q]
4    end
5  end
```

View:

pages/search.html.erb

```
1  <% breadcrumb :search, @term %>
2
3  <% parent_breadcrumb do |link| %>
4    <%= link_to "Back to #{link.text}", link.url %>
5  <% end %>
6
7  <div class="page-header">
8    <h1>Searching for <%= @term %></h1>
9  </div>
10
11 <p>Guess what? Coming soon!</p>
```

And a search form:

layouts/application.html.erb

```

1  [...]
2  <div class="navbar navbar-inverse" role="navigation">
3    <div class="container">
4      <div class="navbar-header">
5        <%= link_to 'Music shop', root_path, class: 'navbar-brand' %>
6      </div>
7      <ul class="nav navbar-nav">
8        <li><%= link_to 'Albums', albums_path %></li>
9      </ul>
10
11     <ul class="nav navbar-nav navbar-right">
12       <li><%= link_to 'Profile', user_path(1) %></li>
13     </ul>
14
15     <%= form_tag search_path, method: :get, class: 'navbar-form navbar-right' do %>
16       <%= text_field_tag 'q', nil, placeholder: 'Search' %>
17     <% end %>
18   </div>
19 </div>
20 [...]

```

Create a new crumb:

breadcrumbs.rb

```

1  [...]
2  crumb :search do |keyword|
3    link "Searching for #{keyword}", search_path(q: keyword)
4  end

```

As you can see, route helpers can be used to generate a link with the actual search term that was provided by the user. Go on and try this out!

By the way, you can also fetch the parent breadcrumb and create a “Go back” link like this:

```

1  <% parent_breadcrumb do |link| %>
2    <%= link_to "Back to #{link.text}", link.url %>
3  <% end %>

```

Which can come in handy sometimes.

The last thing to mention is that one crumb may have multiple links. For example, we could do this:

breadcrumbs.rb

```
1 crumb :root do
2   link "Music shop"
3   link "Home", root_path
4 end
5 [...]
```

Now, there will be a “Music shop” link displayed before the “Home” link.

We can also tweak the link’s name depending on a condition. For example, if our shop has nice discounts on Mondays, we can notify users about that (OK, that is a really contrived example):

breadcrumbs.rb

```
1 crumb :root do
2   link "Music shop" + (Time.now.monday? ? "(SALES!)" : '')
3   link "Home", root_path
4 end
5 [...]
```

Scaling the Trail

For large websites, the *breadcrumbs.rb* file may become really complicated. Fortunately, Gretel allows the configuration to be split into multiple files easily. To do this, create a new directory called *breacrums* inside *config* and place *.rb* files with your config there.

In this demo, I’ve created *albums.rb* with the crumbs related to albums and *other_stuff.rb* with other crumbs. You may remove the *breadcrumbs.rb* file or leave it with some other configuration – it will still be loaded.

Another thing worth mentioning is that breadcrumbs will also be loaded from any engine (<http://guides.rubyonrails.org/engines.html>) your app has, however breadcrumbs in the main app take a higher priority.

Conclusion

This is all for today, folks. We’ve taken a look at the Gretel gem to easily add breadcrumbs to your app. The gem is flexible and really easy to use, so I encourage you to give it a try!

What do you think of breadcrumbs? Are they useful? Have you ever implemented them in your app?

Did you use a gem or did you write everything from scratch? Share your experience!

Thanks for reading!



[\(http://www.sitepoint.com/author/ibodrov/\)](http://www.sitepoint.com/author/ibodrov/)

Ilya Bodrov-Krukowski (<http://www.sitepoint.com/author/ibodrov/>)

Ilya Bodrov is a senior engineer working at Campaigner LLC, teaching assistant at Learnable and lecturer at Russian State Technological University (Internet Technology department). His primary programming languages are Ruby (with Rails) and JavaScript (AngularJS). He enjoys coding, teaching people and learning new things. Ilya also has some Cisco and Microsoft certificates and was working as a tutor in an educational center for a couple of years. In his free time he writes posts for his [website](http://radiant-wind.com) (<http://radiant-wind.com>), participates in OpenSource projects, goes in for sports and plays music.

[🐦 \(https://twitter.com/bodrovis\)](https://twitter.com/bodrovis) [🐙 \(https://github.com/bodrovis\)](https://github.com/bodrovis)


[in \(http://ru.linkedin.com/pub/ilya-bodrov/93/8a6/603\)](http://ru.linkedin.com/pub/ilya-bodrov/93/8a6/603)

[g+ \(https://plus.google.com/103641984440210150447\)](https://plus.google.com/103641984440210150447)

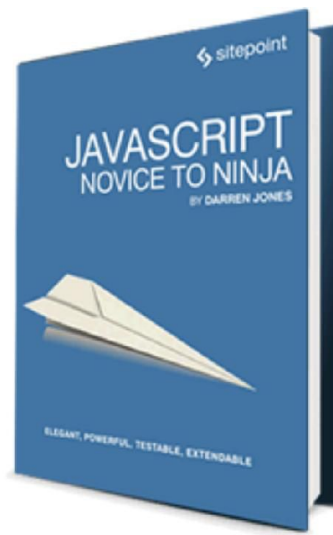
You might also like:

Sprockets Dissected: Rack and Middleware

(<http://www.sitepoint.com/sprockets-dissected-rack-middleware/>) 

Book: Jump Start Rails (https://learnable.com/books/jump-start-rails/?utm_source=sitepoint&utm_medium=related-items&utm_content=getting-started-ruby) 

Sprockets Dissected: Asset Tags (<http://www.sitepoint.com/sprockets-dissected-asset-tags/>) 



Free *JavaScript: Novice to Ninja* Sample

Get a free 32-page chapter of *JavaScript: Novice to Ninja* and receive updates on exclusive offers from SitePoint.

Claim Now

No Reader comments

About

[About us \(/about-us/\)](#)

[Advertise \(/advertising\)](#)

[Press Room \(/press\)](#)

[Legals \(/legals\)](#)

[Feedback \(mailto:feedback@sitepoint.com\)](mailto:feedback@sitepoint.com)

[Write for Us \(/write-for-us\)](#)

Our Sites

[Learnable \(https://learnable.com\)](https://learnable.com)

[Reference \(http://reference.sitepoint.com\)](http://reference.sitepoint.com)

[Web Foundations \(/web-foundations/\)](#)

Connect