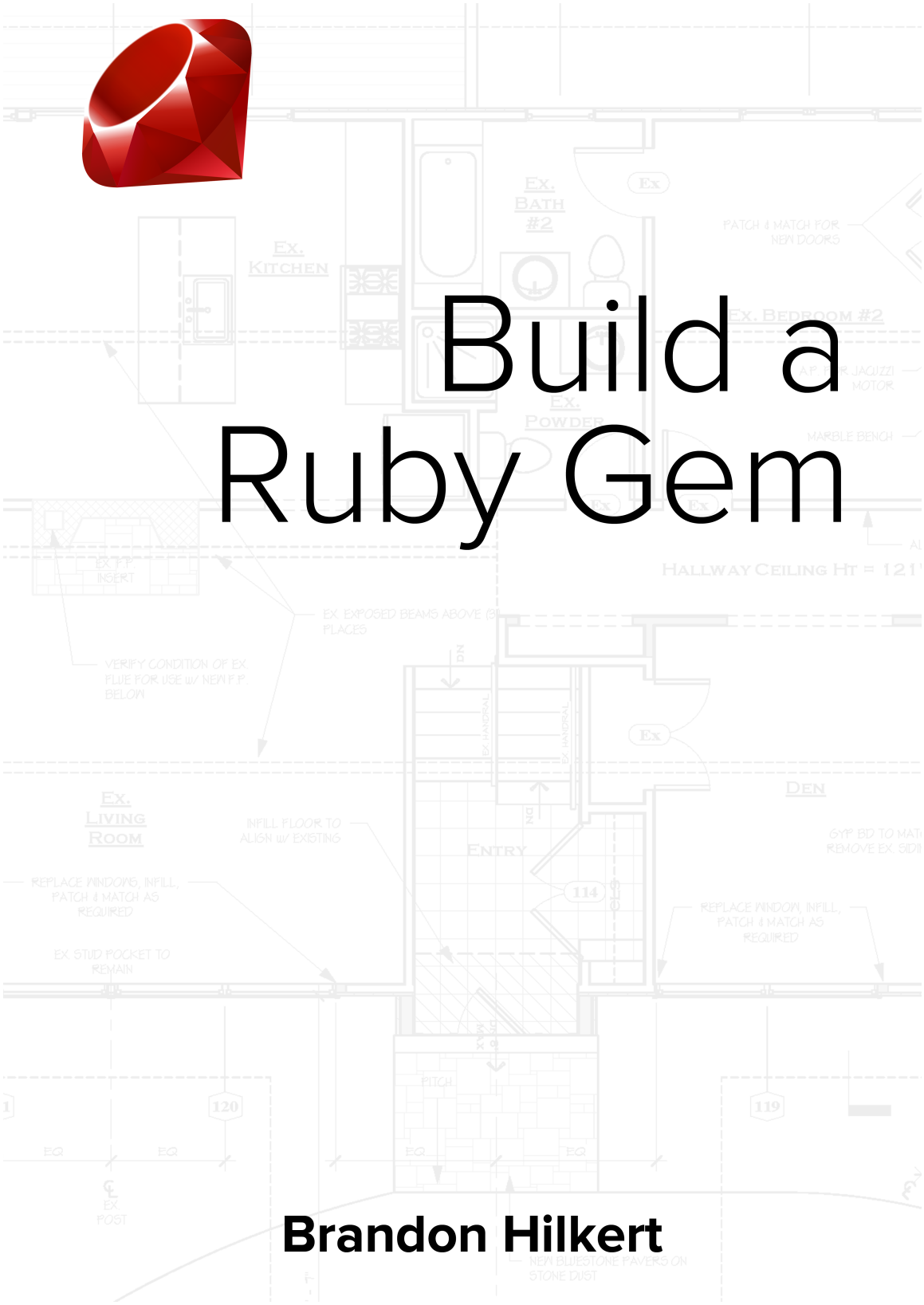

Build a Ruby Gem

Brandon Hilkert

Rails Engines

Rails engines range from simple plugins to powerful micro-applications. The discussions we've had so far about Railties are closely related to the function of a Rails engine. One interesting side note is that a Rails application is a Rails engine itself — so it's easy to see how we can encapsulate just about any normal Rails functionality in an engine, to ultimately embed in a host application.

The Rails engine documentation is well written and touches on the many ways to include functionality. I won't cover every detail of Rails engines in this chapter, just enough to get you started making use of them. It's possible to make full applications (routes, controllers, models, migrations, etc.) using Rails engines. However, we're going to focus on some of the simpler the elements of a Rails engine that allow us to integrate functionality where a Railtie won't suffice. Just know, there is far more you can do with Rails engines than what we'll cover here. The documentation link above provides examples of many of those use cases.

Use Case

I wrote a blog post about rendering relatives timestamps in Rails on the client. Using a client-side solution like timeago.js allows timestamps to update in real-time, so when we see "15 minutes ago", we know the record *really* is 15 minutes old.

The other benefit to using a client-side library like `timeago.js` is timezone offsets are handled automatically. All we have to do is render the UTC time in the view and javascript will handle converting it to the relative timezone of the browser, along with keeping it updated as the page becomes stale.

For this example, we'll make a gem that integrates `timeago.js` in to a Rails application and provides a simple way to render relative timestamps in a view.

The Implementation

The implementation of our gem will require:

1. Adding the `timeago.js` javascript asset to the asset pipeline
2. Adding a javascript asset to invoke the `timeago()` plugin
3. Adding a view helper to render html in a format the javascript plugin expects

The third requirement was covered in a previous chapter on view helpers, so we'll cruise through that one quickly. However, including assets hasn't been covered and it's the perfect segue in to the benefits of using a Rails engine, rather than just a Railtie.

Rails engines allow us create the same directory structure of a Rails application, since a Rails application is just a Rails engine. By indicating our gem is an engine (we'll see how to do this soon…), Rails will pick up the related files in the Rails-specific directories in our gem. So if we add a file in the `app/helpers/` directory of our gem, that same file will be available in the host Rails application. The same approach applies for controllers, models, migrations, assets and anything else we would add to a typical Rails application.

To start, let's create our new gem:

```
$ bundle gem time_ago
      create  time_ago/Gemfile
      create  time_ago/Rakefile
      create  time_ago/LICENSE.txt
      create  time_ago/README.md
      create  time_ago/.gitignore
      create  time_ago/time_ago.gemspec
      create  time_ago/lib/time_ago.rb
      create  time_ago/lib/time_ago/version.rb
Initializing git repo in /Users/bhilkert/Dropbox/code/time_ago
```

When we include javascript libraries from external sources, the `vender/` directory is the most appropriate place for them. Let's create the directory `vendor/javascripts/` and place the source for the `timeago.js` plugin there:

```
├── Gemfile
├── LICENSE.txt
├── Rakefile
├── lib
|   ├── time_ago
|   |   └── version.rb
|   └── time_ago.rb
├── time_ago.gemspec
└── vendor
    └── assets
        └── jquery.timeago.js
```

To create our view helper, we'll add the `/app/helpers/` directory:

```
├── Gemfile
├── LICENSE.txt
├── Rakefile
├── app
|   └── helpers
|       └── time_ago_helper.rb
├── lib
|   ├── time_ago
|   |   └── version.rb
|   └── time_ago.rb
├── time_ago.gemspec
└── vendor
    └── assets
        └── jquery.timeago.js
```

The code for the view helper is shown below:

```
module TimeAgoHelper
  def timeago(time, options = {})
    options[:class] ||= "timeago"
    content_tag(
     :time,
     time.to_s,
     options.merge(datetime: time.getutc.iso8601)
    ) if time
  end
end
```

Lastly, we want to include a javascript asset to invoke the `timeago.js` plugin on page change (this includes the initial page load). This is almost identical to the first step of including the vendored `timeago.js` asset, except we're going to put it in the `app/assets/javascripts/` directory since it's not an external library, but rather a javascript include that will invoke the javascript plugin.

Let's create the directory `app/assets/javascripts/` and place the following file there:

```
// app/assets/javascripts/timeago.js
//
// jQuery Timeago setup for timeago helper
//
//= require jquery.timeago

$(document).on('page:change', function() {
  $("time.timeago").timeago();
});
```

This file serves as both a manifest file for the `jquery.timeago.js` asset and a function to invoke the plugin whenever the page loads or changes.

Lastly, we need to designate our gem as an engine. The default entry file that was created when we used bundler to bootstrap our gem looked like this:

```
require "time_ago/version"

module TimeAgo
end
```

All we need to do is add the `Engine` class and inherit from `Rails::Engine`, giving us:

```
require "time_ago/version"

module TimeAgo
  class Engine < ::Rails::Engine
  end
end
```

At this point, because our gem is so closely tied to Rails, we should add Rails as a dependency in our gemspec:

```
# coding: utf-8
lib = File.expand_path('../lib', __FILE__)
$LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
require 'time_ago/version'

Gem::Specification.new do |spec|
  spec.name          = "time_ago"
  spec.version       = TimeAgo::VERSION
  spec.authors       = ["Brandon Hilkert"]
  spec.email         = ["brandonhilkert@gmail.com"]
  spec.summary       = %q{A gem to integrate the timeago.js}
  spec.description   = %q{A gem to integrate the timeago.js}
  spec.homepage      = ""
  spec.license       = "MIT"

  spec.files         = `git ls-files -z`.split("\x0")
  spec.executables   = spec.files.grep(%r{^bin/}) { |f| File.basename(f) }
  spec.test_files    = spec.files.grep(%r{^(test|spec|features)/})
  spec.require_paths = ["lib"]

  spec.add_dependency "rails", ">= 3.1"

  spec.add_development_dependency "bundler", "~> 1.5"
  spec.add_development_dependency "rake"
end
```

Note: In addition to adding Rails as a dependency, we've also specified that it's only compatible with Rails version `3.1` or later because of the need for the asset pipeline.

Moving to a sample Rails application, we can include the gem in our host application by adding it to the `Gemfile` using the path option:

```
gem "time_ago", path: "../time_ago"
```

Since we included an asset that needs to be included in the Rails asset pipeline, we have to take one more step and instruct the user to add the following to their `app/assets/javascripts/application.js` manifest file:

```
//= require timeago
```

This directive actually refers to the `app/assets/javascripts/timeago.js` file we included in our gem to invoke the timeago.js plugin on page change.

Now when we load our Rails application, tags using the `timeago` view helper get rendered to UI as:

```
<time class="timeago" datetime="2014-01-08T14:55:58Z">
    2014-01-08 14:55:58 UTC
</time>
```

and are updated by the javascript plugin to:

```
<time class="timeago" datetime="2014-01-08T15:04:10Z"
   title="2014-01-08 15:04:10 UTC">
     18 days ago
</time>
```

Implementations in the Wild

One of the greatest examples of making the most of a Rails engine is Devise. Devise is one of the more popular options for adding authentication to a Rails application. Just looking at the app directory of the gem, we can see Devise adds functionality through controllers, helpers, mailers and views. The structure of Devise is fairly complicated because it is doing so much, but here is where the Rails engine is defined allowing the elements in the `/app` directory (among other things) to be integrated in to a Rails application.

The local_time gem from Basecamp is a simple example of using the asset includes of a Rails engine to render relative timestamps. It's similar to the one we created above, but uses the moment.js javascript library instead. Because it's well tested and likely to be supported long-term, I'd suggest using it instead of the gem we created above.

Summary

Prior to Rails 3, Rails integration at this level was not possible. The introduction of Rails engines created a new way to organize micro-applications and integrate them into a host application. Doing so keeps features and otherwise separate logic truly separate.

I've only scratched the surface on what a gem can provide through engines. It's certainly more detailed than just asset and view helper integration, as shown above. If you're

interested in learning more about the other features Rails engine provide, the Rails guides are a great place to start.

In the next and final chapter, we'll explore open source projects and the best ways to get started contributing.