



Installation

Add this line to your application's Gemfile:

```
1 gem 'decent_exposure', '~> 3.0'
```

And then execute:

```
1 $ bundle
```

Or install it yourself as:

```
1 $ gem install decent_exposure
```

API

The whole API consists of three methods so far: `expose`, `expose!`, and `exposure_config`.

In the simplest scenario you'll just use it to expose a model in the controller:

```
1 class ThingsController < ApplicationController
2   expose :thing
3 end
```

Now every time you call `thing` in your controller or view, it will look for an ID and try to perform `Thing.find(id)`. If the ID isn't found, it will call `Thing.new(thing_params)`. The result will be memoized in an `@exposed_thing` instance variable.

Example Controller Here's what a standard Rails 5 CRUD controller using Decent Exposure might look like:

```
1 class ThingsController < ApplicationController
2   expose :things, ->{ Thing.all }
3   expose :thing
4
5   def create
6     if thing.save
7       redirect_to thing_path(thing)
8     else
9       render :new
10    end
11  end
12
13  def update
14    if thing.update(thing_params)
15      redirect_to thing_path(thing)
16    else
17      render :edit
18    end
19  end
20
21  def destroy
22    thing.destroy
23    redirect_to things_path
24  end
25
26  private
27
28  def thing_params
29    params.require(:thing).permit(:foo, :bar)
30  end
31 end
```

Under the Hood

The default resolving workflow is pretty powerful and customizable. It could be expressed with the following pseudocode:

```
1 def fetch(scope, id)
2   instance = id ? find(id, scope) : build(build_params, scope)
3   decorate(instance)
```

```

4  end
5
6  def id
7    params[:thing_id] || params[:id]
8  end
9
10 def find(id, scope)
11   scope.find(id)
12 end
13
14 def build(params, scope)
15   scope.new(params) # Thing.new(params)
16 end
17
18 def scope
19   model # Thing
20 end
21
22 def model
23   exposure_name.classify.constantize # :thing -> Thing
24 end
25
26 def build_params
27   if respond_to?(:thing_params, true) && !request.get?
28     thing_params
29   else
30     {}
31   end
32 end
33
34 def decorate(thing)
35   thing
36 end

```

The exposure is also lazy, which means that it won't do anything until you call the method. To eliminate this laziness you can use the `expose!` macro instead, which will try to resolve the exposure in a before filter.

It is possible to override each step with options. The acceptable options to the `expose` macro are:

fetch

This is the entry point. The `fetch` proc defines how to resolve your exposure in the first place.

```

1  expose :thing, fetch: ->{ get_thing_some_way_or_another }

```

Because the above behavior overrides the normal workflow, all other options would be ignored. However, Decent Exposure is decent enough to actually blow up with an error so you don't accidentally

do this.

There are other less verbose ways to pass the `fetch` block, since you'll probably be using it often:

```
1 expose(:thing){ get_thing_some_way_or_another }
```

Or

```
1 expose :thing, ->{ get_thing_some_way_or_another }
```

Or even shorter

```
1 expose :thing, :get_thing_some_way_or_another
```

There is another shortcut that allows you to redefine the entire fetch block with less code:

```
1 expose :comments, from: :post
2 # equivalent to
3 expose :comments, ->{ post.comments }
```

id

The default fetch logic relies on the presence of an ID. And of course Decent Exposure allows you to specify how exactly you want the ID to be extracted.

Default behavior could be expressed using following code:

```
1 expose :thing, id: ->{ params[:thing_id] || params[:id] }
```

But nothing is stopping you from throwing in any arbitrary code:

```
1 expose :thing, id: ->{ 42 }
```

Passing lambdas might not always be fun, so here are a couple of shortcuts that could help make life easier.

```
1 expose :thing, id: :custom_thing_id
2 # equivalent to
3 expose :thing, id: ->{ params[:custom_thing_id] }
4
5 expose :thing, id: [:try_this_id, :or_maybe_that_id]
6 # equivalent to
7 expose :thing, id: ->{ params[:try_this_id] || params[:or_maybe_that_id]
  ] }
```

find

If an ID was provided, Decent Exposure will try to find the model using it. Default behavior could be expressed with this configuration:

```
1 expose :thing, find: ->(id, scope){ scope.find(id) }
```

Where `scope` is a model scope, like `Thing` or `User.active` or `Post.published`.

Now, if you're using `FriendlyId` or `Stringex` or something similar, you'd have to customize your finding logic. Your code might look somewhat like this:

```
1 expose :thing, find: ->(id, scope){ scope.find_by!(slug: id) }
```

Again, because this is likely to happen a lot, Decent Exposure gives you a decent shortcut so you can get more done by typing less.

```
1 expose :thing, find_by: :slug
```

build

When an ID is not present, Decent Exposure tries to build an object for you. By default, it behaves like this:

```
1 expose :thing, build: ->(thing_params, scope){ scope.new(thing_params)
  }
```

build_params

These options are responsible for calculating params before passing them to the build step. The default behavior was modeled with Strong Parameters in mind and is somewhat smart: it calls the `thing_params` controller method if it's available and the request method is not `GET`. In all other cases it produces an empty hash.

You can easily specify which controller method you want it to call instead of `thing_params`, or just provide your own logic:

```
1 expose :thing, build_params: :custom_thing_params
2 expose :other_thing, build_params: ->{ { foo: "bar" } }
3
4 private
5
6 def custom_thing_params
7   # strong parameters stuff goes here
```

8 **end**

scope

Defines the scope that's used in `find` and `build` steps.

```
1 expose :thing, scope: ->{ current_user.things }
2 expose :user, scope: ->{ User.active }
3 expose :post, scope: ->{ Post.published }
```

Like before, shortcuts are there to make you happier:

```
1 expose :post, scope: :published
2 # equivalent to
3 expose :post, scope: ->{ Post.published }
```

and

```
1 expose :thing, parent: :current_user
2 # equivalent to:
3 expose :thing, scope: ->{ current_user.things }
```

model

Allows you to specify the model class to use. Pretty straightforward.

```
1 expose :thing, model: ->{ AnotherThing }
2 expose :thing, model: AnotherThing
3 expose :thing, model: "AnotherThing"
4 expose :thing, model: :another_thing
```

decorate

Before returning the thing, Decent Exposure will run it through the decoration process. Initially, this does nothing, but you can obviously change that:

```
1 expose :things, ->{ Thing.all.map{ |thing| ThingDecorator.new(thing) }
  }
2 expose :thing, decorate: ->(thing){ ThingDecorator.new(thing) }
```

exposure_config

You can pre-save some configuration with `exposure_config` method to reuse it later.

```
1 exposure_config :cool_find, find: ->{ very_cool_find_code }
2 exposure_config :cool_build, build: ->{ very_cool_build_code }
3
4 expose :thing, with: [:cool_find, :cool_build]
5 expose :another_thing, with: :cool_build
```

Rails Mailers

Mailers and Controllers use the same `decent_exposure` dsl.

Example Mailer

```
1 class PostMailer < ApplicationMailer
2   expose(:posts, -> { Post.last(10) })
3   expose(:post)
4
5   def top_posts
6     @greeting = "Top Posts"
7     mail to: "to@example.org"
8   end
9
10  def featured_post(id:)
11    @greeting = "Featured Post"
12    mail to: "to@example.org"
13  end
14 end
```

Rails Scaffold Templates

If you want to generate rails scaffold templates prepared for `decent_exposure` run:

```
1 rails generate decent_exposure:scaffold_templates [--template_engine
  erb|haml]
```

This will create the templates in your `lib/templates` folder.

Make sure you have configured your templates engine for generators in `config/application.rb`:

```
1 # config/application.rb
2 config.generators do |g|
3   g.template_engine :erb
4 end
```

Now you can run scaffold like:

```
1 rails generate scaffold post title description:text
```

Contributing

1. Fork it (https://github.com/hashrocket/decent_exposure/fork)
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Commit your changes (`git commit -am 'Add some feature'`)
4. Push to the branch (`git push origin my-new-feature`)
5. Create a new Pull Request

About



Decent Exposure is supported by the team at Hashrocket, a multidisciplinary design & development consultancy. If you'd like to work with us or join our team, don't hesitate to get in touch.