

Ruby® on Rails

Notes for Professionals

Chapter 6: Rails Best Practices

Section 6.1: Fat Model, Skinny Controller

"Fat Model, Skinny Controller" refers to how the M and C parts of MVC ideally work together. A response-related logic should go in the model. Ideally in a rich, testable method. Meanwhile, it is simply a nice interface between the view and model.

In practice, this can require a range of different types of refactoring, but it all comes down to any logic that isn't about the response to the model (instead of the controller), not only how where possible but you've also made it possible to test your code outside of the context of

Let's look at a simple example. Say you have code like this:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at > ?', Time.now)
end
```

You can change it to this:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Then, you can move the logic to your post model, where it might look like this:

```
scope :published, => { where('published_at <= ?', Time.now) }
scope :unpublished, => { where('published_at > ?', Time.now) }
```

Section 6.2: Domain Objects (No More Fat Model)

"Fat Model, Skinny Controller" is a very good first step, but it doesn't scale well. Let's think on the **Simple Responsibility** of models. What is the single response logic it is to hold non-response-related logic?

No. Its responsibility is to handle the persistence layer and its abstractions. Business logic as well as any non-response-related logic and non-persistence objects.

Domain objects are classes designed to have only one responsibility in the application. "Scream their architecture" for the problems they solve.

In practice, you should strive towards skinny models, slowly view and solution shouldn't be influenced by the framework you're choosing.

For example

Let's say you're a marketplace which charges a fixed 15% commission to your customers via Stripe. If you charge a fixed 15% commission, that means that your commission changes depending on the order's amount because Stripe

Ruby® on Rails Notes for Professionals

Chapter 12: ActiveRecord Validations

Section 12.1: Validating length of an attribute

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 3 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..30 }
  validates :registration_number, length: { in: 8 }
end
```

The possible length constraint options are:

- minimum** - The attribute cannot have less than the specified length.
- maximum** - The attribute cannot have more than the specified length.
- in** (or **within**) - The attribute length must be included in a given interval. The value for this option must be a range.
- is** - The attribute length must be equal to the given value.

Section 12.2: Validates format of an attribute

Validate that an attribute's value matches a regular expression using `format` and the `with` option.

```
class User < ApplicationRecord
  validates :name, format: { with: /^[A-Za-z]+$/ }
end
```

You can also define a constant and set its value to a regular expression and pass it to the `with` option. This might be more convenient for really complex regular expressions.

```
PHONE_REGEX = /^[A-Za-z0-9]{3}-[0-9]{3}$/
validates :phone, format: { with: PHONE_REGEX }
```

The default error message is `is invalid`. This can be changed with the `message` option.

```
validates :bio, format: { with: /^[A-Za-z]+$/, message: "Numbers are not allowed" }
```

The reverse also applies, and you can specify that a value should not match a regular expression with the `without` option.

Section 12.3: Validating presence of an attribute

This helper validates that the specified attributes are not empty.

```
class Person < ApplicationRecord
  validates :name, presence: true
end
```

```
Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

You can use the `absence` helper to validate that the specified attributes are absent. It uses the `presence?` check for all or empty values.

Ruby® on Rails Notes for Professionals

Chapter 42: Decorator pattern

Section 42.1: Decorating a Model using Draper

Draper automatically matches up models with their decorators by convention.

```
# ApplicationController/Decorator.rb
class UserDecorator < Draper::Decorator
  attr_reader :user
  attr :full_name

  def initialize(subject)
    @subject = subject
  end

  def created_at
    Time.utc.strftime(current_user.time_zone) do
      object.created_at.strftime("%a, %d %b %Y %I:%M %p")
    end
  end
end
```

Given a `Decorator` variable containing an ActiveRecord object, you can access your decorator by calling `decorator` on the `Decorator`, or by specifying the Draper class if you want to be specific.

```
# user = user.decorator || Draper::Decorator.new(user)
#> user.full_name
#> user.created_at
```

Section 42.2: Decorating a Model using SimpleDelegator

Most Rails developers start by modifying their model information within the template itself:

```
<%=> <%= user.first_name %> |<%= user.last_name %>| %w/%s %s/
#> user.created_at.in_time_zone(current_user.time_zone).strftime("%a, %d %b %Y %I:%M %p")
```

For models with a lot of data, this can quickly become cumbersome and lead to copy-pasting logic from one template to another.

This example uses `SimpleDelegator` from the `stdlib`.

All requests to a `SimpleDelegator` object are passed to the parent object by default. You can override any method with presentation logic, or you can add new methods that are specific to this view.

`SimpleDelegator` provides two methods: `delegate`... to set what object is being delegated to, and `__getobj__`... to get that object.

```
class UserDecorator < SimpleDelegator
  attr_reader :user
  def initialize(user, view)
    __setobj__(user, view)
  end
end

# new methods can call methods on the parent explicitly
def full_name
  ["first_name", "last_name"]
end
```

Ruby® on Rails Notes for Professionals

200+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Ruby on Rails	2
Section 1.1: Creating a Ruby on Rails Application	2
Section 1.2: Create a new Rails app with your choice of database and including the RSpec Testing Tool	4
Section 1.3: Generating A Controller	4
Section 1.4: Installing Rails	5
Section 1.5: Create a new Rails app with a non-standard database adapter	7
Section 1.6: Creating Rails APIs in JSON	7
Section 1.7: Generate a Resource with Scaffolds	8
Chapter 2: Routing	10
Section 2.1: Resource Routing (Basic)	10
Section 2.2: Constraints	11
Section 2.3: Scoping routes	13
Section 2.4: Concerns	15
Section 2.5: Root route	16
Section 2.6: Split routes into multiple files	16
Section 2.7: Additional RESTful actions	17
Section 2.8: Member and Collection Routes	17
Section 2.9: Mount another application	18
Section 2.10: Nested Routes	18
Section 2.11: Redirection	19
Section 2.12: Redirects and Wildcard Routes	19
Section 2.13: Scope available locales	19
Section 2.14: URL params with a period	20
Chapter 3: ActiveRecord	21
Section 3.1: Creating a Model via generator	21
Section 3.2: Introduction to Callbacks	21
Section 3.3: Creating a Model manually	22
Section 3.4: Manually Testing Your Models	23
Section 3.5: Creating A Migration	23
Section 3.6: Create a Join Table using Migrations	24
Section 3.7: Using a model instance to update a row	25
Chapter 4: Views	26
Section 4.1: Structure	26
Section 4.2: Partials	26
Section 4.3: AssetTagHelper	27
Section 4.4: Replace HTML code in Views	28
Section 4.5: HAML - an alternative way to use in your views	29
Chapter 5: ActiveRecord Migrations	31
Section 5.1: Adding multiple columns to a table	31
Section 5.2: Add a reference column to a table	31
Section 5.3: Rollback migrations	32
Section 5.4: Add a new column with an index	32
Section 5.5: Run specific migration	33
Section 5.6: Redo migrations	33
Section 5.7: Add a new column to a table	33

Section 5.8: Remove an existing column from a table	34
Section 5.9: Add column with default value	34
Section 5.10: Running migrations in different environments	35
Section 5.11: Create a new table	35
Section 5.12: Running migrations	35
Section 5.13: Change an existing column's type	35
Section 5.14: Create a hstore column	36
Section 5.15: Create a join table	36
Section 5.16: Add a self reference	37
Section 5.17: Create an array column	37
Section 5.18: Add an unique column to a table	37
Section 5.19: Checking migration status	38
Section 5.20: Changing Tables	38
Section 5.21: Adding a NOT NULL constraint to existing data	38
Section 5.22: Forbid null values	39
Chapter 6: Rails Best Practices	40
Section 6.1: Fat Model, Skinny Controller	40
Section 6.2: Domain Objects (No More Fat Models)	40
Section 6.3: Beware of default_scope	42
Section 6.4: Convention Over Configuration	44
Section 6.5: Don't Repeat Yourself (DRY)	45
Section 6.6: You Ain't Gonna Need it (YAGNI)	45
Chapter 7: Naming Conventions	47
Section 7.1: Controllers	47
Section 7.2: Models	47
Section 7.3: Filenames and autoloading	47
Section 7.4: Views and Layouts	48
Section 7.5: Models class from Controller name	48
Chapter 8: ActionCable	49
Section 8.1: User Authentication	49
Section 8.2: [Basic] Server Side	49
Section 8.3: [Basic] Client Side (Coffeescript)	49
Chapter 9: ActiveRecord	51
Section 9.1: Using ActiveRecord::Validations	51
Chapter 10: User Authentication in Rails	52
Section 10.1: Authentication using Devise	52
Section 10.2: Devise Controller Filters & Helpers	52
Section 10.3: OmniAuth	53
Section 10.4: has_secure_password	53
Section 10.5: has_secure_token	53
Chapter 11: ActiveRecord Associations	55
Section 11.1: Polymorphic association	55
Section 11.2: Self-Referential Association	55
Section 11.3: belongs_to	56
Section 11.4: has_one	56
Section 11.5: has_many	56
Section 11.6: The has_many :through association	57
Section 11.7: The has_one :through association	57
Section 11.8: The has_and_belongs_to_many association	58

Chapter 12: ActiveRecord Validations	59
Section 12.1: Validating length of an attribute	59
Section 12.2: Validates format of an attribute	59
Section 12.3: Validating presence of an attribute	59
Section 12.4: Custom validations	60
Section 12.5: Validates inclusion of an attribute	60
Section 12.6: Grouping validation	61
Section 12.7: Validating numericality of an attribute	61
Section 12.8: Validate uniqueness of an attribute	62
Section 12.9: Skipping Validations	62
Section 12.10: Confirmation of attribute	62
Section 12.11: Using :on option	63
Section 12.12: Conditional validation	63
Chapter 13: ActiveRecord Query Interface	64
Section 13.1: .where	64
Section 13.2: .where with an array	64
Section 13.3: Scopes	65
Section 13.4: Get first and last record	66
Section 13.5: Ordering	67
Section 13.6: .where.not	67
Section 13.7: Includes	68
Section 13.8: Joins	68
Section 13.9: Limit and Offset	69
Section 13.10: .find_by	69
Section 13.11: .delete_all	69
Section 13.12: ActiveRecord case insensitive search	69
Section 13.13: .group and .count	70
Section 13.14: .distinct (or .uniq)	70
Chapter 14: ActionMailer	71
Section 14.1: Basic Mailer	71
Section 14.2: Generating a new mailer	72
Section 14.3: ActionMailer Interceptor	72
Section 14.4: Adding Attachments	72
Section 14.5: ActionMailer Callbacks	73
Section 14.6: Generate a Scheduled Newsletter	73
Chapter 15: Rails generate commands	79
Section 15.1: Rails Generate Controller	79
Section 15.2: Rails Generate Migration	80
Section 15.3: Rails Generate Scaffold	80
Section 15.4: Rails Generate Model	81
Chapter 16: Configuration	82
Section 16.1: Custom configuration	82
Chapter 17: I18n - Internationalization	84
Section 17.1: I18n with arguments	84
Section 17.2: Translating ActiveRecord model attributes	84
Section 17.3: Get locale from HTTP request	86
Section 17.4: Pluralization	87
Section 17.5: Set locale through requests	87
Section 17.6: Use I18n with HTML Tags and Symbols	89
Section 17.7: Use I18n in views	89

Chapter 18: Using GoogleMaps with Rails	91
Section 18.1: Add the google maps javascript tag to the layout header	91
Section 18.2: Geocode the model	91
Section 18.3: Show addresses on a google map in the profile view	92
Section 18.4: Set the markers on the map with javascript	93
Section 18.5: Initialize the map using a coffee script class	94
Section 18.6: Initialize the map markers using a coffee script class	94
Section 18.7: Auto-zoom a map using a coffee script class	95
Section 18.8: Exposing the model properties as json	96
Chapter 19: File Uploads	98
Section 19.1: Single file upload using Carrierwave	98
Section 19.2: Nested model - multiple uploads	98
Chapter 20: Caching	100
Section 20.1: Russian Doll Caching	100
Section 20.2: SQL Caching	100
Section 20.3: Action caching	101
Section 20.4: Fragment caching	101
Section 20.5: Page caching	102
Section 20.6: HTTP caching	102
Chapter 21: ActionController	103
Section 21.1: Basic REST Controller	103
Section 21.2: Filters	104
Section 21.3: Generating a controller	106
Section 21.4: Rescuing ActiveRecord::RecordNotFound with redirect_to	107
Section 21.5: Display error pages for exceptions	107
Section 21.6: Output JSON instead of HTML	108
Section 21.7: Controllers (Basic)	108
Section 21.8: Parameters	109
Section 21.9: Filtering parameters (Basic)	109
Section 21.10: Redirecting	110
Section 21.11: Using Views	110
Chapter 22: Configuration	112
Section 22.1: Rails General Configuration	112
Section 22.2: Configuring assets	112
Section 22.3: Configuring generators	112
Section 22.4: Environments in Rails	113
Section 22.5: Database Configuration	113
Chapter 23: Safe Constantize	115
Section 23.1: Successful safe_constantize	115
Section 23.2: Unsuccessful safe_constantize	115
Chapter 24: Rails 5	116
Section 24.1: How to install Ruby on Rails 5 on RVM	116
Section 24.2: Creating a Ruby on Rails 5 API	116
Chapter 25: Authorization with CanCan	119
Section 25.1: Getting started with CanCan	119
Section 25.2: Handling large number of abilities	119
Section 25.3: Defining abilities	121
Section 25.4: Quickly test an ability	121
Chapter 26: Mongoid	123

[Click here to download full PDF material](#)