[**AmenZhou**](https://github.com/AmenZhou)**/**[**ruby\_on\_rails\_learning**](https://github.com/AmenZhou/ruby_on_rails_learning)

Zombie Rails 2

Level 1

Database variable types in rails string text integer boolean decimal float binary date time datetime

Migration option default: value limit: 30 null: false first: true after: :email unique: true

Add or remove columns to tables

rails g migration add<columns>to<table> name:value

rails g migration Remove<Anything>From<Table name> name:value

Migration commands

rename\_column :table, :old\_column, :new\_column

rename\_table :old\_table, :new\_table

drop\_table :table

change\_column :table, :column, :type, :option

change\_column\_default :table, :column, default: true

remove\_column :table, :column

add\_column :table, :column, :type, :option

rake

**rake db:setup -- create db, load schema, run seed**

Level 2

1 Write a scope on the Tweet model called recent which returns the 4 most recent tweets. Hint: You'll need an order AND a limit scope.

class Tweet < ActiveRecord::Base

scope :recent, order("created\_at desc").limit(4)

end

2 Write another scope called graveyard which only shows tweets where the show\_location column is true and the location is "graveyard"

class Tweet < ActiveRecord::Base

scope :recent, order('created\_at desc').limit(4)

scope :graveyard, where(show\_location: true, location: 'graveyard')

end

3 In this controller action create an instance variable called @graveyard\_tweets which uses both of the two scopes recent and graveyard together.

class TweetsController < ApplicationController

def index

@tweets = Tweet.all

@graveyard\_tweets = Tweet.recent.graveyard

end

end

4 Create a before\_save callback that checks to see if a tweet has a location, if it does have a location then set show\_location to true. Tip: You can check to see if location exists with if self.location?

class Tweet < ActiveRecord::Base

before\_save :set\_show\_location

def set\_show\_location

self.show\_location = true if self.location? #self.location this self is optional

end

end

5 Add callbacks so the appropriate log function is called after an update and destroy.

class Tweet < ActiveRecord::Base

after\_update :log\_update

after\_destroy :log\_destroy

def log\_update

logger.info "Tweet #{id} updated"

end

def log\_destroy

logger.info "Tweet #{id} deleted"

end

end

6 Instead of storing location inside the Tweet model, let's instead break it out into a separate table (as you see below). In this case we want to define that a Tweet can have one Location, and a Location belongs to a Tweet. Fill out the models below accordingly.

class Tweet < ActiveRecord::Base

has\_one :location

end

class Location < ActiveRecord::Base

belongs\_to :tweet

end

\*\*\*\*\*\*\*\*\*\*\*\*\****Pay attendtion of foreign\_key***\*\*\*\*\*\*\*\*\*\* 7 OH NO! Our Database Admin turned into a Zombie and decided to rename the belongs\_to field in our locations table tweeter\_id instead of the intelligent default tweet\_id. We're going to slay him and correct this, but in the meantime set the foreign\_key on both relationships to tweeter\_id. Also set the dependency so when a tweet is destroyed, the location is destroyed as well.

class Tweet < ActiveRecord::Base

has\_one :location, dependent: :destroy, foreign\_key: :tweeter\_id

end

class Location < ActiveRecord::Base

belongs\_to :tweet, foreign\_key: :tweeter\_id

end

8 We're going to be iterating through many tweets and printing out their location. Refactor the controller code below to use the includes method.

class TweetsController < ApplicationController

def index

@tweets = Tweet.recent.includes(:location).all

end

end

9 A Tweet can belong to one or more Categories (e.g. eating flesh, walking dead, searching for brains). Write a migration that creates two tables, categories, and categorizations. Give categories one column named name of type string; and give categorizations two integer columns: tweet\_id and category\_id.

class AddTweetCategories < ActiveRecord::Migration

def change

create\_table :categories do |t|

t.string :name

end

create\_table :categorizations do |t|

t.integer :tweet\_id

t.integer :category\_id

end

end

end

10 Now that we have our new tables, it's time to define the relationships between each of the models. Define the has\_many through relationships in the Tweet & Category model and the belongs\_to relationships in the Categorization model.

class Tweet < ActiveRecord::Base

has\_many :categorizations

has\_many :categories, through: :categorizations

end

class Categorization < ActiveRecord::Base

belongs\_to :tweet

belongs\_to :category

end

class Category < ActiveRecord::Base

has\_many :categorizations

has\_many :tweets, through: :categorizations

end

Level 3

radio button

<%= f.radio\_button :decomp, 'fresh', checked: true %>

select

<%= f.select :decomp, [['fresh', 1], ['rotting', 2], ['stale', 3]]%>

test input helper

<%= f.password\_field :password %>

<%= f.number\_field :price %>

<%= f.range\_field :quantity %>

Nested routes pic-- Code School - Rails for Zombies Two-nested routes

Challenges

2 Create the form for entering tweet status (text\_area) and location (text\_field) using the appropriate Rails view helpers. All you need is a form\_for block, the input helpers, and a submit button.

<h1>New tweet</h1>

<%= form\_for(@tweet) do |f| %>

<%= f.text\_area :status %>

<%= f.text\_field :location %>

<%= f.submit value='Submit' %>

<% end %>

3 Look at the following database table and create the proper input fields for the columns listed here.

<%= form\_for(@weapon) do |f| %>

<%= f.text\_field :name %>

<%= f.number\_field :ammo %>

<%= f.check\_box :is\_broken %>

<% end %>

4 Rather than having a weapon that is broken or not, lets instead have a condition field which is either "New", "Rusty", or "Broken". Add a set of radio buttons where the user can select one of these states. Make "New" be checked by default.

<%= form\_for(@weapon) do |f| %>

<%= f.text\_field :name %>

<%= f.number\_field :ammo %>

<%= f.radio\_button :condition, 'New', checked: true %>

<%= f.radio\_button :condition, 'Rusty' %>

<%= f.radio\_button :condition, 'Broken' %>

<% end %>

5 Instead of using radio buttons, use a select box for the condition. Refactor the code below:

<%= form\_for(@weapon) do |f| %>

<%= f.select :condition, ['New', 'Rusty', 'Broken'] %>

<% end %>

6 Write the nested route that will allow us to nest tweets and weapons under the zombie resource. The idea here is that a zombie has many tweets and zombie has many weapons.

RailsForZombies::Application.routes.draw do

resources :zombies do

resources :tweets

resources :weapons

end

end

7 Now that we have the proper route, we need to make sure the weapons controller properly looks up both the zombie and the weapon when we request /zombies/2/weapons/1. Finish this controller:

class WeaponsController < ApplicationController

def show

@zombie = Zombie.find(2)

@weapon = @zombie.weapons.find(1)

end

end

8 Now create the proper link\_to for when we view a zombie and want to show each of its weapons, and when we want to create a new weapon for this zombie.

<h2><%= @zombie.name %>'s weapons</h2>

<ul>

<% @weapons.each do |w| %>

<li><%= link\_to w.name, [@zombie, w] %></li>

<% end %>

</ul>

<%= link\_to "New Weapon", new\_zombie\_weapon\_path(@zombie) %>

9 Change the form\_for below to use the proper nesting for creating a new weapon for a Zombie.

<%= form\_for([@zombie, @weapon]) do |f| %>

<%= f.text\_field :name %>

<%= f.submit %>

<% end %>

10 Modify the following code to make it more pretty, using titleize, to\_sentence, pluralize, and number\_to\_currency (in just that order)

<h2><%= @zombie.name.titleize %></h2>

<p>Weapons: <%= @zombie.weapon\_list.to\_sentence %></p>

<p><%= pluralize(@zombie.tweets.size, 'Tweet') %></p>

<p>Money in Pocket <%= number\_to\_currency(@zombie.money) %></p>

11 Refactor the code below to move the form into the \_form.html.erb partial.

<h2>New Tweet</h2>

<%= render 'form' %>

<%= link\_to 'back', tweets\_path %>

Level 5

1 Enter the command for generating a mailer called WeaponMailer which has the emails low\_ammo and broken.

rails g mailer WeaponMailer low\_ammo broken

2 Code up the low\_ammo mailer with the subject of "#{weapon.name} has low ammo", the email should be sent to the zombie.email. Lastly, set the default from address for all emails in WeaponMailer to [admin@rfz.com](mailto:admin@rfz.com)

class WeaponMailer < ActionMailer::Base

default from: "admin@rfz.com"

def low\_ammo(weapon, zombie)

mail to: zombie.email, subject: "#{weapon.name} has low ammo"

end

end

3 Finish coding the check\_ammo method on the Weapon model so when we have exactly three ammo left, it will send out the low\_ammo mailer we just created.

class Weapon < ActiveRecord::Base

belongs\_to :zombie

before\_save :check\_ammo

def check\_ammo

if ammo == 3

WeaponMailer.low\_ammo(self, self.zombie).deliver

end

end

end

4 Change the low\_ammo method to include a picture of the weapon that's low on ammo as an attachment. You can name the file weapon.jpg and load the file using weapon.picture\_file.

class WeaponMailer < ActionMailer::Base

default from: "admin@rfz.com"

def low\_ammo(weapon, zombie)

attachments['weapon.jpg'] = weapon.picture\_file

mail to: zombie.email, subject: "#{weapon.name} has low ammo"

end

end

5 Convert the following to their appropriate asset tags.

<%= javascript\_include\_tag "weapon" %>

<%= image\_tag "weapon.png" %>

<%= stylesheet\_link\_tag "weapon" %>

6 Convert the following scss.erb file to properly reference the asset\_path for the image listed in it. Also, try refactoring the scss to use nesting.

h2#newUser {

text-indent: -9999px;

a {

height: 64px;

width: 50px;

display: block;

background: url(<%= asset\_path('rails.png') %>) no-repeat;

}

}

7 Use CoffeeScript so when the New Weapon link is pressed it makes the #newWeapon div visible and then hides the New Weapon link. Don't forget to call preventDefault().

$(document).ready ->

$('#displayWeaponForm').click (event) ->

event.preventDefault()

$(this).hide()

$('#newWeapon').show()

Level 5

1 Complete the method below so that if the ammo is low it will render the fire\_and\_reload view, otherwise it should render the fire\_weapon view.

class WeaponsController < ApplicationController

def fire\_weapon

@weapon = Weapon.find(params[:id])

@weapon.fire!

if @weapon.low\_ammo?

render 'fire\_and\_reload'

end

end

end

2 Create two custom member routes on the weapons resource, so you have a put method called toggle\_condition and a put method called reload.

RailsForZombies::Application.routes.draw do

resources :zombies do

resources :weapons do

put :toggle\_condition, on: :member

put :reload, on: :member

end

end

end

3 Complete the create method below. When @weapon.save is successful it should render the @weapon object in JSON, have status :created, and set the location to the @weapon's show url. When @weapon.save fails it should return the @weapon.errors and have the status :unprocessable\_entity.

class WeaponsController < ApplicationController

def create

@weapon = Weapon.new(params[:weapon])

if @weapon.save

render json: @weapon, status: :created, location: @weapon

else

render json: @weapon.errors, status: :unprocessable\_entity

end

end

end

4 Complete the controller so that it returns in JSON only the amount of ammo which is left in the weapon. If the ammo has less than 30 bullets it should return the status code :ok, and if not it should return the status code :unprocessable\_entity.

class WeaponsController < ApplicationController

def reload

@weapon = Weapon.find(params[:id])

if @weapon.ammo < 30

@weapon.reload(params[:ammo\_to\_reload])

render json: @weapon.to\_json(only: :ammo), status: :ok

else

render json: @weapon.to\_json(only: :ammo), status: :unprocessable\_entity

end

end

end

5 Modify the show action so that the JSON it renders includes the zombie record the @weapon belongs to. Also make it exclude the :id, :created\_at, and :updated\_at fields.

class WeaponsController < ApplicationController

def show

@weapon = Weapon.find(params[:id])

render json: @weapon.to\_json(include: :zombie, except: [:id, :created\_at, :updated\_at])

end

end

6 Edit the as\_json method so the Zombie class only returns the zombie's name and weapons (use include). Only return the weapon's name and ammo.

class Zombie < ActiveRecord::Base

has\_many :weapons

def as\_json(options=nil)

super(options ||

{only: :name, include: :weapons, only: [:name, :ammo]})

end

end

7 Modify the show.html.erb view below so that both the Toggle link and the Reload form use AJAX. All you need to do is add the option that makes them ajaxified.

<ul>

<li>

<em>Name:</em> <%= @weapon.name %>

</li>

<li>

<em>Condition:</em>

<span id="condition"><%= @weapon.condition %></span>

<%= link\_to "Toggle", toggle\_condition\_weapon\_path(@weapon), remote: true %>

</li>

<li>

<em>Ammo:</em>

<span id="ammo"><%= @weapon.ammo %></span>

</li>

</ul>

<%= form\_for @weapon, url: reload\_weapon\_path(@weapon), remote: true do |f| %>

<div class="field">

Number of bullets to reload:

<%= number\_field\_tag :ammo\_to\_reload, 30 %> <br /> <%= f.submit "Reload" %>

</div>

<% end %>

8 Modify the toggle\_condition action so that it responds to JavaScript, and complete the toggle\_condition.js.erb using jQuery to update the condition span with the @weapon's changed condition and make it highlight.

$('span#condition').append("<%= escape\_javascript(@weapon.condition) %>").effect('highlight');

9 Now write the controller and JavaScript code needed to properly reload the weapon using the ajaxified form. In the reload.js.erb use jQuery to update the #ammo text to the current @weapon.ammo value and if the ammo value is over or equal to 30, fadeOut the #reload\_form div.

$('span#ammo').text("<%= @weapon.ammo %>");

<% if @weapon.ammo >= 30 %>

$('div#reload\_form').fadeOut();

<% end %>

10 Instead of returning jQuery which gets executed on the client-side, lets write the ajax request in CoffeeScript communicating with JSON. It should do the same thing as the last challenge, updating & highlighting the ammo, and fading out the form (hint: fade out the wrapper element) if ammo is equal or above 30. Tip for your ajax form: data: {ammo\_to\_reload: ammo}.

$(document).ready ->

$('div#reload\_form form').submit (event) ->

event.preventDefault()

url = $(this).attr('action')

ammo = $('#ammo\_to\_reload').val()

$.ajax

type: 'put'

url: url

data: {weapon: {ammo: ammo} }

dataType: 'json'

success: (json) ->

$('#ammo').text(json.ammo).effect('highlight')

$('#reload\_form').fadeOut() if json.ammo >= 0

ScreenCast

Asset Pipline

bundle open jquery-rails rails s -e production rake assets:precompile

Ruby Bits

Level 2

2 We want to make sure that each game is a valid game object - in this case a simple hash of values. Even still, we wouldn't want to return a hash with a nil name. Raise an InvalidGameError error in the new\_game method if name is nil.

class InvalidGameError < StandardError; end

def new\_game(name, options={})

raise InvalidGameError, "You must provide a name" if name.nil?

{

name: name,

year: options[:year],

system: options[:system]

}

end

begin

game = new\_game(nil)

rescue InvalidGameError => e

puts "There was a problem creating your new game: #{e.message}"

end

3 When passing in an array of arguments to a method, sometimes it'll make sense to use Ruby's "splat" operator rather than explicitly requesting an array. Update the describe\_favorites method and the call to it to instead use the splat operator.

def describe\_favorites(\*games)

for game in games

puts "Favorite Game: #{game}"

end

end

describe\_favorites('Mario', 'Contra', 'Metroid')

4 Passing around hashes is getting troublesome, let's use a class to hold our data. We've started the Game class for you, now please implement the initialize method to store name, system and year in instance variables.

class Game

def initialize(name, options={})

@name=name

@year=options[:year]

@system=options[:system]

end

end

6 attr\_accessor Whoever created the game object will want to be able to access the name, year and system for the game, but that doesn't mean we need to make getter methods for them. Refactor the code below to make these variables available using the Ruby way with attr\_accessor.

class Game

attr\_accessor :name, :year, :system

def initialize(name, options={})

@name = name

@year = options[:year]

@system = options[:system]

end

end

7 When a game is initialized, store another variable called created\_at which is set to Time.now in the initialize method. Make sure it can be accessed, but that it cannot be set from outside the object.

class Game

attr\_accessor :name, :year, :system

attr\_reader :created\_at

def initialize(name, options={})

@name = name

@year = options[:year]

@system = options[:system]

@created\_at = Time.new

end

end

level 3

1

class Library

attr\_accessor :games

def initialize(games)

@games = games

end

end

2

class Library

attr\_accessor :games

def initialize(games)

self.games = games

end

def has\_game?(game)

for game in games

return true if game == @game

end

false

end

end

3 We can initialize our Library with an array of games, but the only way to add games from outside the class is to use the games accessor method and alter the array. This is breaking encapsulation, so let's create a new method in Library called add\_game which takes in a game and adds it to the games array.

class Library

attr\_accessor :games

def add\_game(game)

self.games << game

end

def initialize(games)

self.games = games

end

def has\_game?(search\_game)

for game in games

return true if game == search\_game

end

false

end

end

4 Things are looking good! We're able to use our Library class to store our games now. Whenever we call add\_game, let's call a new private method called log which will call puts with some information about the game that was added. Your log method should take in a string to be displayed.

class Library

attr\_accessor :games

def initialize(games)

self.games = games

end

def has\_game?(search\_game)

for game in games

return true if game == search\_game

end

false

end

def add\_game(game)

self.games << game

log("The game name is #{game.name}, it was built in #{game.year}, for #{game.system}")

end

private

def log(message)

puts message

end

end

5

class ArcadeGame < Game

end

class ConsoleGame < Game

end

6 Inheritance II For our ArcadeGame class, we'll also want to track the weight of these giant cabinets taking up all of our available space. Luckily we thought ahead: we already take in an options parameter that we can stick weight into! Override the initialize method for ArcadeGame to take in the same parameters as its parent class, call super, and then set weight.

class ArcadeGame < Game

attr\_accessor :weight

def initialize(name, options={})

super

self.weight = options[:weight]

end

end

class ConsoleGame < Game

end

7 Inheritance III Whenever we output a game right now it'll show up using the to\_s method from Object, the parent object of Game. A basic to\_s implementation is completed below on Game. Override this for ConsoleGame to also show the system the game is on.

class ConsoleGame < Game

def to\_s

super + self.system

end

end

8 Refactoring Our to\_s method will come in very handy. Whenever we need to output a game, rather than calling a method on the game, we can just output the game object and Ruby will call to\_s on it automatically. Refactor both classes below. Change the description method of Game to use the to\_s method implicitly. Then remove any duplicated code in ConsoleGame. Note: you'll need to use self inside a class to reference the entire object.

Original

class Game

attr\_accessor :name, :year, :system

attr\_reader :created\_at

def initialize(name, options={})

self.name = name

self.year = options[:year]

self.system = options[:system]

@created\_at = Time.now

end

def to\_s

self.name

end

def description

"#{self.name} was released in #{self.year}."

end

end

class ConsoleGame < Game

def to\_s

"#{self.name} - #{self.system}"

end

def description

"#{self.name} - #{self.system} was released in #{self.year}."

end

end

Changed

class Game

attr\_accessor :name, :year, :system

attr\_reader :created\_at

def initialize(name, options={})

self.name = name

self.year = options[:year]

self.system = options[:system]

@created\_at = Time.now

end

def to\_s

self.name

end

def description

"#{self} was released in #{self.year}."

end

end

class ConsoleGame < Game

def to\_s

"#{super} - #{self.system}"

end

end

Level 4

1 print array from Contra to the end of games

def last\_games(games, index) games.from(index) end games = ["Super Mario Bros.", "Contra", "Metroid", "Mega Man 2"] index = games.index('Contra') puts last\_games(games, index)

3 20 years after game release

def anniversary(game, years) game[:release].advance(years: years) end

game = { name: 'Contra', release: DateTime.new(1987, 2, 20, 0, 0, 0) } puts anniversary(game, 20)

4 Using ActiveSupport, return the difference between Mario's favorite games and Luigis's favorite games by implementing the difference\_between method.

def difference\_between(player1, player2) player1.diff(player2) end

mario\_favorite = { sports: "Mario Sports Mix", action: "Super Mario World" }

luigi\_favorite = { sports: "Golf", action: "Super Mario World" }

puts difference\_between(mario\_favorite, luigi\_favorite)

5 Implement the exclude\_character method below to return characters except the passed in character. Use ActiveSupport to return these key/pair values. Also, change the call to exclude\_character so that yoshi's games are excluded.

def exclude\_character(games, character) games.except(character) end

games = { mario: ["Super Mario World", "Super Smash Bros. Melee"], luigi: ["Luigi's Mansion"], yoshi: ["Yoshi's Island", "Yoshi's Story"] } puts exclude\_character(games, :yoshi)

6 Numbers Refactor the describe\_count method below to use ActiveSupport in order to find out if a number is even or odd.

def describe\_count(games) if games.empty? "You have no games" elsif games.length.even? "You have an even number of games" elsif games.length.odd? "You have an odd number of games" end end

games = ["Super Mario Bros.", "Contra", "Metroid", "Mega Man 2"] puts describe\_count(games)

7 Strings Implement the convert\_title method to use one of String's core extension methods. Given the input below, this method should return the string 'Super Mario Bros.'

def convert\_title(title) title.titleize end

puts convert\_title("super mario bros.")

Ruby Bites Lesson 5

1 Create a module named GameUtils and place the lend\_to\_friend method inside the module. Change lend\_to\_friend to a class method by prefixing it with self.. You won't need to require this module since it'll be inside the same file (already required), but you will have to namespace your method call.

module GameUtils def self.lend\_to\_friend(game, friend\_email) end end

game = Game.new("Contra") GameUtils.lend\_to\_friend(game, "[gregg@codeschool.com](mailto:gregg@codeschool.com)")

2 Re-open the Game class and include the GameUtils module so its methods are exposed as instance methods. Make sure to do this before it is called.

class Game include GameUtils def initialize(name) @name = name end end

game = Game.new("contra") game.lend\_to\_friend("Gregg")

3 Good job! Now expose the methods from the GameUtils module as class methods of the Game class.

class Game extend GameUtils

end

Game.find\_all\_from\_user("Gregg")

4 Object Extend Extend the single game object with the Playable module, so we can call the play method on it.

game = Game.new("Contra") game.extend(Playable) game.play

5 Hook Methods Define a new self.included method hook for the LibraryUtils module which will extend the ClassMethods on the passed in class. Also, since we'll now be extending ClassMethods when LibraryUtils is included, remove duplicate code in the AtariLibrary class.

module LibraryUtils def self.included(base) base.extend(ClassMethods) end

def add\_game(game) end

def remove\_game(game) end

module ClassMethods def search\_by\_game\_name(name) end end end

class AtariLibrary include LibraryUtils end

6 ActiveSupport::Concern - Part I Now refactor the following code to use ActiveSupport::Concern's ability to expose class methods from a module.

module LibraryUtils

extend ActiveSupport::Concern

def add\_game(game) end

def remove\_game(game) end

module ClassMethods def search\_by\_game\_name(name) end end end

7 Call the included method from inside the LibraryUtils module and pass in a block that calls the load\_game\_list class method.

module LibraryUtils

extend ActiveSupport::Concern

included do

load\_game\_list

end

def add\_game(game)

end

def remove\_game(game)

end

module ClassMethods

def search\_by\_game\_name(name)

end

def load\_game\_list

end

end

end

8 Make sure the AtariLibrary class includes only the LibraryUtils module and let ActiveSupport::Concern take care of loading its dependencies. Then, refactor the self.included method on LibraryUtils to use the included method.

module LibraryLoader

extend ActiveSupport::Concern

module ClassMethods

def load\_game\_list

end

end

end

module LibraryUtils

extend ActiveSupport::Concern

include LibraryLoader

included do

load\_game\_list

end

end

class AtariLibrary

include LibraryUtils

end

**Ruby Bites Level 6 -- Block**

**1 Let's build a Library class that will manage our growing collection of games. We've already written a list method that prints the names of all our games, but it uses an ugly for loop to iterate the list. Refactor it to use each with a block instead.**

class Library

attr\_accessor :games

def initialize(games = [])

self.games = games

end

def list

games.each do |game|

puts game.name

end

end

end

**2 We'd like to be able to operate on our games by system. Implement an each\_on\_system method that iterates over our games using each and yields to a block for every game on the requested system. To test that it's working, we'll call each\_on\_system with a simple block that prints a message for every Super Nintendo game in our library. See the example.rb below.**

class Library

attr\_accessor :games

def initialize(games = [])

self.games = games

end

def each\_on\_system(system)

games.each {|game| yield if game.system == system}

end

end

**3 Our each\_on\_system method is working, but it's not very useful unless the block has access to each game that we find. Modify each\_on\_system to pass the Game object into the block so we can print its name.**

class Library

attr\_accessor :games

def initialize(games = [])

self.games = games

end

def each\_on\_system(system)

games.each do |game|

yield(game) if game.system == system

end

end

end

**4 Earlier we wrote a list method that prints the name of each game in our library. We can make the output formatting more flexible by allowing a block to be passed to the list method. We'll yield each game to the block and allow the block to format and return a string for us to display. Modify the list method to yield to a block and print whatever the block returns.**

class Library

attr\_accessor :games

def initialize(games = [])

self.games = games

end

def list

games.each do |game|

puts yield(game)

end

end

end

**5 Using Enumerable**

**Let's add the power of Ruby's Enumerable module to our game library. Implement an each method that yields each game in the library. Finally, include the Enumerable module so that we'll be able to call methods like select and collect on our library.**

class Library

attr\_accessor :games

include Enumerable

def initialize(games = [])

self.games = games

end

def each

games.each do |game|

yield game

end

end

end

**6 Refactoring with Blocks**

**Now that our library is complete, let's play some games! A friend has given us his Emulator class to use, and we've implemented methods to play a game and grab a screenshot. But look at all that duplicated code in play and screenshot. Refactor the duplication (the begin, new and rescue parts) into a private method called emulate that handles the emulator setup and exception handling and yields the emulator instance to a block.**

class Game

attr\_accessor :name, :year, :system

attr\_reader :created\_at

def initialize(name, options={})

self.name = name

self.year = options[:year]

self.system = options[:system]

@created\_at = Time.now

end

def play

emulate {|emulator| emulator.play(self)}

end

def screenshot

emulate {|emulator| emulator.start(self); emulator.screenshot}

end

private

def emulate

begin

emulator = Emulator.new(system)

yield(emulator)

rescue Exception => e

puts "Emulator failed: #{e}"

end

end

end

**Test Level 1**

1 Write a basic conditional test using assert which checks if 1 > 0. Name your test class ConditionalTest.

require 'test/unit'

class ConditionalTest < Test::Unit::TestCase

def test\_true\_or\_false

assert 1 > 0

end

end

2 Add the custom error message One is not greater than zero to the failing assertion we just created

class ConditionalTest < Test::Unit::TestCase

def test\_one\_greater\_than\_zero

assert 0 > 1, "one is not greater than zero"

end

end

3 In a moment we are going to create the multiple\_of? in our Multiple module which returns true if a number is a multiple of another number. Finish the test below, asserting that Multiple.multiple\_of?(10, 5) returns true.

class MultipleTest < Test::Unit::TestCase

def test\_multiple\_of

assert Multiple.multiple\_of?(10, 5)

end

end

4 Now that we have a failing test, let's make it pass by creating the self.multiple\_of?(multiple, num) method in the Multiple module. Hint: one way of checking multiples is using the modulo operator, multiple % num == 0 (will be 'true' if multiple can be divided evenly by the num)

module Multiple

def self.multiple\_of?(x, y)

x % y == 0

end

end

5 We are going to create a Zombifier class with a zombify method that upcases and adds 'BRAINS'. Lets begin by writing a test using assert\_not\_nil to make sure zombify returns something.

class ZombifierTest < Test::Unit::TestCase

def test\_zombify\_returns\_something

z = Zombifier.new('make me a zombie')

assert\_not\_nil z.zombify

end

end

6 Notice our zombifier.rb file and how the zombify method adds 'BRAINS' to the string, use assert\_match to test if zombify is doing this correctly.

class ZombifierTest < Test::Unit::TestCase

def test\_zombify\_brains

z = Zombifier.new('I love your arms')

assert\_match /BRAINS/, z.zombify

end

end

7 Now, using assert\_equal, write a test to make sure zombify returns the expected string, upcase with 'BRAINS'. ex. "HELLO WORLD BRAINS" is expected when we call Zombifier.new("Hello world").zombify

class ZombifierTest < Test::Unit::TestCase

def test\_zombify\_upcase

z = Zombifier.new("Hello world")

assert\_equal "HELLO WORLD BRAINS", z.zombify

end

end

8 Notice in zombifier.rb how we now raise a RuntimeError if the string already looks like a zombie (contains 'BRAINS'). Test for this behavior using assert\_raise.

class ZombifierTest < Test::Unit::TestCase

def test\_brains\_in\_zombify\_raises\_error

z = Zombifier.new('BRAINS')

assert\_raise(RuntimeError){z.zombify}

end

end

9 Since zombify is supposed to modify an existing string, it should also return a string. Create a test using assert\_kind\_of to make sure a String is being returned.

class ZombifierTest < Test::Unit::TestCase

def test\_zombify\_returns\_a\_string

z = Zombifier.new('I like knees')

assert\_kind\_of String, z.zombify

end

end

**Testing level 2**

1 Using an assert and the valid? method, test that a 'tweet' is not valid without a status.

class TweetTest < ActiveSupport::TestCase

test "invalid without a status" do

tweet = Tweet.new

assert !tweet.valid?

end

end

2 Execute the rake command which will run both db:test:prepare and all the tests.

rake db:test:prepare

3 Lets try another validation test. This time, test to make sure a tweet is valid with all its attributes before save. A tweet has a zombie and a status (you'll need to create a zombie for this).

class TweetTest < ActiveSupport::TestCase

test "valid with all attributes" do

zombie = Zombie.new

tweet = Tweet.new

tweet.status = 'Hello world'

tweet.zombie = zombie

assert tweet.valid?

end

end

4 Create a tweets fixture in the tweets.yml file. The Tweet model has a zombie\_id that's an Integer and a status that's a String.

hello\_world:

zombie\_id: 1

status: "hello\_world"

5 Now that we have fixtures tweets.yml and zombies.ymlbelow, let's clean up some tests. Add fixtures to the following tests.

class TweetTest < ActiveSupport::TestCase

test "valid with all attributes" do

z = zombies(:ash)

t = tweets(:hello\_world)

assert t.valid?, "tweet isn't valid"

end

end

**notice the 'zombies(:ash)' this method name is same as yml file name zombies.yml**

6 Create a test that ensures the brains? method returns true if a status contains 'brains'.

class TweetTest < ActiveSupport::TestCase

test "can detect brains" do

tweet = tweets(:hello\_world)

tweet.status = 'brains'

assert tweet.brains?

end

end

7 Create a test to ensure that the hello\_world tweet contains zombie Ash.

class TweetTest < ActiveSupport::TestCase

test "contains a zombie" do

t = tweets(:hello\_world)

z = zombies(:ash)

z.tweets.all? {|t| t.zombie == z }

assert t.zombie == z

end

end

**Level 3**

1 SETUP METHOD <http://rtfz.codeschool.com/levels/3/challenges/1>

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "invalid without a status" do

@tweet.status = nil

assert !@tweet.valid?, "Status is not being Validated"

end

test "valid with all attributes" do

assert @tweet.valid?, "tweet isn't valid"

end

end

2 CUSTOM ASSERT

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

# Don't change this, use it to refactor the tests below.

def assert\_attribute\_is\_validated(model, attribute)

# This line sets the specified attribute to nil

model.assign\_attributes(attribute => nil)

assert !model.valid?, "#{attribute.to\_s} is not being validated"

end

test "invalid without a status" do

assert\_attribute\_is\_validated(@tweet, :status)

end

test "invalid without a zombie" do

assert\_attribute\_is\_validated(@tweet, :zombie)

end

end

3 INTRODUCING SHOULDA

<http://rtfz.codeschool.com/levels/3/challenges/4>

class TweetTest < ActiveSupport::TestCase

should validate\_presence\_of(:status)

should validate\_presence\_of(:zombie)

end

4 SHOULDA I

<http://rtfz.codeschool.com/levels/3/challenges/5>

class TweetTest < ActiveSupport::TestCase

should validate\_uniqueness\_of(:id)

should validate\_numericality\_of(:id)

end

5 SHOULDA II <http://rtfz.codeschool.com/levels/3/challenges/6>

class TweetTest < ActiveSupport::TestCase

should ensure\_length\_of(:status).is\_at\_least(3).is\_at\_most(140)

end

**Level 4**

1 STUBBING

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "show\_author\_summary should set status to zombie summary" do

@tweet.zombie.stubs(:zombie\_summary)

@tweet.show\_author\_summary

assert\_equal @tweet.zombie.zombie\_summary, @tweet.status, 'tweet status does not contain zombie summary'

end

end

2 MOCKING

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "show\_author\_summary should call zombie\_summary" do

@tweet.zombie.expects(:zombie\_summary)

@tweet.show\_author\_summary

end

end

3 STUB + ASSERT

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "status\_image calls the ZwitPic get\_status\_image api" do

ZwitPic.expects(:get\_status\_image).with(@tweet.id)

@tweet.status\_image

end

end

4 STUBS & MOCKING 

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "status\_image calls the ZwitPic get\_status\_image api" do

ZwitPic.expects(:get\_status\_image).with(@tweet.id).returns(["name.png", 'http://eathead.com'])

@tweet.status\_image

end

end

5 RETURNING PROPER RESULTS

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "status\_image returns a properly formated HTML image element with alt and src" do

ZwitPic.stubs(:get\_status\_image).with(@tweet.id).returns({url: 'eatbrain.com', name: 'pic.png'})

assert\_equal "<img src='eatbrain.com' alt='pic.png' />", @tweet.status\_image

end

end

6 OBJECT STUB

class TweetTest < ActiveSupport::TestCase

def setup

@tweet = tweets(:hello\_world)

end

test "status\_image returns a properly formated HTML image element with alt and src" do

image = stub({name: 'Yummy brain I ate last night', url: 'http://zwitpic.com/2.jpg'})

ZwitPic.stubs(:get\_status\_image).returns(image)

assert\_equal "<img src='http://zwitpic.com/2.jpg' alt='Yummy brain I ate last night' />", @tweet.status\_image

end

end

**JQuery**

Leve 3

3.1 Refactoring

$(document).ready(function(){

$("button").on("click", function(){

var tour = $(this).closest(".tour");

var discount = tour.data("discount");

var message = $("<span>Call 1-555-jquery-air for a $" + discount + " discount.</span>");

tour.append(message);

$(this).remove();

});

});

3.2 Better On Handlers

$(document).ready(function(){

$(".tour").on("click", "button", function(){

var tour = $(this).closest(".tour");

var discount = tour.data("discount");

var message = $("<span>Call 1-555-jquery-air for a $" + discount + " discount.</span>");

tour.append(message);

$(this).remove();

});

});

3.21 New Filter

$(document).ready(function(){

//Create the click handler here

$("#filters").on('click', ".on-sale", function(){});

});

3.22 New Filter

$(document).ready(function(){

$("#filters").on("click", ".on-sale", function(){

$(".tour").filter(".on-sale").addClass("highlight");

});

});

3.23 New Filter III

$(document).ready(function(){

$("#filters").on("click", ".on-sale", function(){

$(".highlight").removeClass("highlight");

$(".tour").filter(".on-sale").addClass("highlight");

});

$("#filters").on("click", ".featured", function(){

$(".highlight").removeClass("highlight");

$(".tour").filter(".featured").addClass("highlight");

});

});

**Test Level 5**

1 RAILS INTEGRATION TEST

class TweetDisplaysStatusTest < ActionDispatch::IntegrationTest

def setup

@tweet = tweets(:hello\_world)

end

test "Tweet page responds successfully" do

get tweet\_url(@tweet)

assert\_response :success

end

test "Tweet displays status in heading" do

get tweet\_url(@tweet)

assert\_select "h1", @tweet.status

end

end

2 RAILS INTEGRATION TEST - POST

class CreatingATweetTest < ActionDispatch::IntegrationTest

def setup

@zombie = zombies(:ash)

@tweet\_attributes = {tweet: {zombie\_id: @zombie.id, status: 'Test tweet'}}

end

test "Responds with a redirect to the tweet page" do

post tweets\_url, @tweet\_attributes

@tweet = Tweet.last

#notice tweet\_path, test failed in this part

assert\_redirected\_to tweet\_path(@tweet)

end

end

3 CAPYBARA INTEGRATION TEST

class TweetDisplaysStatusTest < ActionDispatch::IntegrationTest

def setup

@tweet = tweets(:hello\_world)

end

test "Tweet displays status in heading" do

visit tweet\_url(@tweet)

within("h1") do

assert has\_content?(@tweet.status)

end

#get tweet\_url(@tweet)

#assert\_select 'h1', @tweet.status

end

end

4 CAPYBARA INTEGRATION TEST II

class CreatingATweetTest < ActionDispatch::IntegrationTest

test 'should create a new tweet' do

visit new\_tweet\_url

fill\_in "tweet\_status", with: 'Looking for brain'

select "Ash", from: "tweet\_zombie\_id"

click\_button "Create Tweet"

assert\_equal tweet\_path(Tweet.last), current\_path

end

end

5 CAPYBARA INTEGRATION TEST III

class CreatingATweetTest < ActionDispatch::IntegrationTest

test 'should go to new tweet page' do

visit root\_path

click\_link "New Tweet"

assert\_equal new\_tweet\_path, current\_path

end

end

6 HELPER METHODS

test/test\_helper.rb

class ActiveSupport::TestCase

def create\_tweet\_for(zombie, status)

visit new\_tweet\_url

fill\_in 'Status', with: status

select zombie, from: 'Zombie'

click\_button 'Create Tweet'

end

end

test/integration/creating\_a\_new\_tweet\_test.rb

class CreatingATweetTest < ActionDispatch::IntegrationTest

test 'should create a new tweet' do

# visit new\_tweet\_url

# fill\_in 'Status', with: 'I love the way your brain feels'

# select 'Ash', from: 'Zombie'

# click\_button 'Create Tweet'

create\_tweet\_for("Ash", "I love the way your brain feels")

assert\_equal tweet\_path(Tweet.last), current\_path

end

end

**Try JQuery Level 4**

4.5 Slide Effect

$(document).ready(function() {

//alert($(".photos").length)

$("#tour").on("click", "button", function() {

$(".photos").slideDown();

});

});

4.6 Slide Effect II

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$(".photos").slideToggle();

//$(".photos").slideDown();

});

});

4.9 Mouseover II

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$(".photos").slideToggle();

});

$(".photos").on("mouseenter", "li", function() {

$(this).find("span").slideToggle();

});

});

4.11 Named Functions

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$(".photos").slideToggle();

});

$(".photos").on("mouseenter", "li", showPhotos).on("mouseleave", "li", showPhotos);

});

function showPhotos(){

$(this).find("span").slideToggle();

}

4.14 Keyup Event Handler

$(document).ready(function() {

$("#nights").on("keyup", function() {

var days = $(this).val();

$("#nights-count").text(days);

});

});

4.15 Keyup Event Handler II

$(document).ready(function() {

$("#nights").on("keyup", function() {

$("#nights-count").text($(this).val());

var total = +$(this).val() \* +$(this).closest(".tour").data("daily-price");

$("#total").text(total);

});

});

4.16 Another Event Handler

$(document).ready(function() {

$("#nights").on("keyup", function() {

var nights = +$(this).val();

var dailyPrice = +$(this).closest(".tour").data("daily-price");

$("#total").text(nights \* dailyPrice);

$("#nights-count").text($(this).val());

});

$("#nights").on("focus", function(){

$(this).val("7");

});

});

4.19 Link Events II

$(document).ready(function() {

$(".see-photos").on("click", function() {

$(this).closest(".tour").find(".photos").slideToggle();

});

});

4.20 Event Parameter

$(document).ready(function() {

$(".see-photos").on("click", function(event) {

event.stopPropagation();

$(this).closest(".tour").find(".photos").slideToggle();

});

$(".tour").on("click", function() {

alert("This should not be called");

});

});

4.21 Event Parameter II

$(document).ready(function() {

$(".see-photos").on("click", function(event) {

event.stopPropagation();

event.preventDefault();

$(this).closest(".tour").find(".photos").slideToggle();

});

$(".tour").on("click", function() {

alert("This should not be called");

});

});

**Level 5**

5.3 CSS I

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).css("background-color", "#252b30");

});

});

5.4 CSS II

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).css({"background-color": "#252b30", "font-weight": "bold"});

});

});

5.5 Show Photo

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).css({"background-color": "#252b30", "font-weight": "bold"});

$(this).find(".photos").show();

});

});

5.6 Refactoring to CSS

//don't add . inside the addClass and removeClass, addClass(".highlight") => wrong

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).addClass("highlight");

$(this).find(".photos").show();

});

$(".tour").on("mouseleave", function(){

$(this).removeClass("highlight");

});

});

5.8 Animate I

$(document).ready(function() {

$('.tour').on('mouseenter', function() {

$(this).addClass('highlight');

//the per night is default transparent , set the opacity to show it

$(this).find(".per-night").animate({opacity: 1});

});

$('.tour').on('mouseleave', function() {

$(this).removeClass('highlight');

});

});

5.9 Animate II

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).addClass("highlight");

$(this).find(".per-night").animate({"opacity": "1", "top": "-14px"});

});

$(".tour").on("mouseleave", function() {

$(this).removeClass("highlight");

});

});

5.10 Animation Speed

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).addClass("highlight");

//add fast arg

$(this).find(".per-night").animate({"top": "-14px","opacity": "1"}, "fast");

});

$(".tour").on("mouseleave", function() {

$(this).removeClass("highlight");

});

});

5.11 Animate III

$(document).ready(function() {

$(".tour").on("mouseenter", function() {

$(this).addClass("highlight");

$(this).find(".per-night").animate({"top": "-14px","opacity": "1"}, "fast");

});

$(".tour").on("mouseleave", function() {

$(this).removeClass("highlight");

//fadeOut when mouse leave

$(this).find(".per-night").animate({top: 0, opacity: 0}, "fast");

});

});

**Test Level 6**

1 FACTORIES

FactoryGirl.define do

factory :zombie do

name 'Sally'

graveyard 'Valley Dim'

end

end

2 COMPLEX FACTORIES

FactoryGirl.define do

factory :zombie do

name 'Ash'

graveyard 'Petrosville'

# Add sally and moe here

factory :sally do

name 'Sally'

graveyard 'Valley Dim'

end

factory :moe do

name 'Moe'

graveyard 'Petrosville'

end

end

end

3 UNIQUE ATTRIBUTES

FactoryGirl.define do

factory :zombie do

sequence(:name) {|i| "Ash#{i}"}

sequence(:graveyard) {|i| "Petrosville Cemetary#{i}"}

end

end

4 ASSOCIATIONS

FactoryGirl.define do

factory :tweet do

status "Dead"

association :zombie

end

end

5 USING FACTORIES

class TweetTest < ActiveSupport::TestCase

test "A tweet requires a status" do

tweet = FactoryGirl.build(:tweet, status: nil)

assert !tweet.valid?, "tweet isn't valid"

end

end

6 USING FACTORIES II

class TweetTest < ActionDispatch::IntegrationTest

test "tweet page has zombie link" do

@tweet = Factory(:tweet)

visit tweets\_url

click\_link @tweet.status

within("h3") do

assert has\_content?(@tweet.zombie.name)

end

end

end

**JQuery The return flight**

1.3 Ajax

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$.ajax("/photos.html",

{

success: function (response){

alert("Ajax");

}

}

);

});

});

1.4 Ajax with Response

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$.ajax('/photos.html', {

success: function(response){

$(".photos").html(response).fadeIn();

}

});

});

});

1.5 $.get Shorthand

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$.get('/photos.html', function(response) {

$('.photos').html(response).fadeIn();

}

);

});

});

1.6 Ajax Data

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$.ajax('/photos.html', {

success: function(response) {

$('.photos').html(response).fadeIn();

},

data: {"location": $("#tour").data("location")}

});

});

});

1.8 Ajax with Errors

$(document).ready(function() {

var el = $("#tour");

el.on("click", "button", function() {

$.ajax('/photos.html', {

data: {location: el.data('location')},

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function(request, errorType, errorMessage){

$(".photos").html("<li>Error: " + errorType + " with message: " + errorMessage + "</li>");

}

});

});

});

1.9 Setting a Timeout

$(document).ready(function() {

var el = $("#tour");

el.on("click", "button", function() {

$.ajax('/photos.html', {

data: {location: el.data('location')},

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function() {

$('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000

});

});

});

1.10 More Ajax Callbacks

$(document).ready(function() {

$("#tour").on("click", "button", function() {

$.ajax('/photos.html', {

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function() {

$('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000,

beforeSend: function(){

$("#tour").addClass("is-fetching");

},

complete: function(){

$("#tour").removeClass("is-fetching");

}

});

});

});

1.11 Event Delegation

$(document).ready(function() {

function showPhotos() {

$(this).find('span').slideToggle();

}

$('.photos').on('mouseenter', 'li', showPhotos)

.on('mouseleave', 'li', showPhotos);

var el = $("#tour");

el.on("click", "button", function() {

$.ajax('/photos.html', {

data: {location: el.data('location')},

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function() {

$('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000,

beforeSend: function() {

$('#tour').addClass('is-fetching');

},

complete: function() {

$('#tour').removeClass('is-fetching');

}

});

});

});

**The Return Flight Level 2**

2.3 Refactor To Objects

var tour = {

init: function(){

$("#tour").on("click", "button", function() {

$.ajax('/photos.html', {

data: {location: $("#tour").data('location')},

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function() {

$('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000,

beforeSend: function() {

$('#tour').addClass('is-fetching');

},

complete: function() {

$('#tour').removeClass('is-fetching');

}

});

});

}

};

$(document).ready(function() {

tour.init();

});

2.4 Event Handler Refactor

var tour = {

init: function() {

$("#tour").on("click", "button", this.fetchPhotos);

},

fetchPhotos: function(){

$.ajax('/photos.html',

{

data: {location: $("#tour").data('location')},

success: function(response) {

$('.photos').html(response).fadeIn();

},

error: function() {

$('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000,

beforeSend: function() {

$('#tour').addClass('is-fetching');

},

complete: function() {

$('#tour').removeClass('is-fetching');

}

}

);

}

}

$(document).ready(function() {

tour.init();

});

2.6 Creating an Function

//notice the capital T of the function name

function Tour() {

console.log("A new Tour was created");

}

$(document).ready(function() {

//new + object name

var tour = new Tour();

});

2.7 Functionality in Functions

function Tour(price) {

console.log("A new Tour was created");

this.price = price;

this.cost = function(nights){

console.log(nights \* this.price);

}

}

$(document).ready(function() {

var tour = new Tour(100);

tour.cost(4);

});

2.8

function Tour(el) {

console.log(el)

}

$(document).ready(function() {

var paris = new Tour($('#paris'));

});

2.9 Function Setup

function Tour(el) {

this.el = el;

//fetchPhotos must be defined before calling

this.fetchPhotos = function(){

alert("hello");

}

this.el.on('click', 'button', this.fetchPhotos);

}

$(document).ready(function() {

var paris = new Tour($('#paris'));

});

2.10 fetchPhotos Refactor

function Tour(el) {

var tour = this;

this.el = el;

this.fetchPhotos = function() {

$.ajax('/photos.html', {

context: tour,

//notice this time it should use el which from Tour parameter, not this.el

//the context only affect success and error functions

data: {location: el.data('location')},

success: function(response) {

this.el.find('.photos').html(response).fadeIn();

},

error: function() {

this.el.find('.photos').html('<li>There was a problem fetching the latest photos. Please try again.</li>');

},

timeout: 3000,

beforeSend: function() {

this.el.addClass('is-fetching');

},

complete: function() {

this.el.removeClass('is-fetching');

}

});

};

this.el.on('click', 'button', this.fetchPhotos);

}

$(document).ready(function() {

var paris = new Tour($('#paris'));

});

**Rails Outlaw**

Allow all http methods on routes

ZombieOutlaws::Application.routes.draw do

match '/undeads', to: 'undeads#index', via: :all

end

Update the following controller test to use the new HTTP verb for partial updates.

class WeaponsControllerTest < ActionController::TestCase

test "updates weapon" do

patch :update, zombie\_id: @zombie, weapons: { name: 'Scythe' }

assert\_redirected\_to zombie\_url(@zombie)

end

end

Routes Concern

ZombieOutlaws::Application.routes.draw do

concern :defensible do

resources :shotguns

resources :rifles

resources :knives

end

resources :sheriffs, concerns: :defensible

resources :gunslingers, concerns: :defensible

resources :preachers, concerns: :defensible

end

Routes Concern

ZombieOutlaws::Application.routes.draw do

concern :defensible do |options|

resources :shotguns, options

resources :rifles, options

resources :knives, options

end

resources :sheriffs, concerns: :defensible

resources :gunslingers, concerns: :defensible

resources :preachers do

concerns :defensible, except: :destroy

end

end

Thread Safety

ZombieOutlaws::Application.configure do

# config.threadsafe!

#prevents class reloading between requests and ensures Rack::Lock is not included in middleware stack

config.cache\_classes = true

#loads all code before new threads are created

config.eager\_load = true

end

**Rails Outlaw**

@zombie = Zombie.find\_or\_create\_by(name: params[:name])

Scope

scope :recent, ->{where('killed\_on > ?', 2.days.ago)}

scope :outlaws, ->{where(status: 'outlaw')}

Return a null

class Deputy < ActiveRecord::Base

def self.zombie\_counterforce

if Zombie.at\_large\_count.zero?

Deputy.none

else

Deputy.where(status: 'available')

end

end

end

Relation

Zombie.where.not(status: 'outlaw')

Zombie.order(name: :desc, killed\_on: :desc)

Weapon.includes(:zombies)

.where("zombies.name = 'Ash'").references(:zombies)

Weapon.includes(:zombies).where(zombies: {name: 'Ash'})

Need a ajax call authentication

module ZombieOutlaws

class Application < Rails::Application

config.time\_zone = 'Central Time (US & Canada)'

config.action\_view.embed\_authenticity\_token\_in\_remote\_forms = true

end

end

Add a new flash type In the controller

add\_flash\_types :groan

redirect\_to @zombie, groan: @zombie.groan

Show flash on the view

<div id='groan'><%= groan %></div>

Collection radio button

<%= form\_for(@limb) do |f| %>

<div class="field">

<%= f.label :zombie %><br />

<%= collection\_radio\_buttons(:limb,

:zombie\_id,

@zombies,

:id,

:name) %>

</div>

<% end %>

Collection check box button

<%= form\_for(@zombie) do |f| %>

<div class="field">

<%= f.label :limbs %><br />

<%= collection\_check\_boxes(:limb,

:id,

@limbs,

:id,

:kind) %>

</div>

<% end %>

Date Type Input

<%= form\_for(@limb) do |f| %>

<div class="field">

<%= f.label :date\_found %><br />

<%= f.date\_field :date\_found %>

</div>

<% end %>

JSON

limbs\_hashes = @limbs.map do |limb|

{ owner\_name: limb.zombie.name, kind: limb.kind }

end

# TODO: Output JSON version of limbs\_hashes.

limbs\_hashes.to\_json

Skip Test

class ZombieTest < ActiveSupport::TestCase

test "should match number of Items" do

skip

zombie = Zombie.new

zombie.items.new(name: 'revolver')

zombie.items.new(name: 'noose')

assert\_equal 2, zombie.item\_count

end

end

Rake Test

rake test:models

Rake Test Verbose

rake test:models TEST\_OPTS="--verbose"

Fresh When

class MostWantedController < ApplicationController

def show

@zombie = Zombie.most\_wanted

fresh\_when(@zombie)

end

end

class MostWantedController < ApplicationController

etag { current\_user.country }

def show

@zombie = Zombie.most\_wanted

fresh\_when(@zombie)

end

def edit

@zombie = Zombie.most\_wanted

fresh\_when(@zombie)

end

end

Cache

<% cache zombie do %>

<li><%= zombie %></li>

<% end %>

class Zombie < ActiveRecord::Base

belongs\_to :weapon, touch: true

end

<% cache @weapon do %>

<section>

<h3><%= @weapon.name %></h3>

<ul>

<%= render partial: 'zombies/zombie',

collection: @weapon.zombies.recent %>

</ul>

<%= link\_to 'Details', @weapon %>

</section>

<% end %>

Live

class SightingsController < ApplicationController

include ActionController::Live

def alerts

response.stream.write "Half-eaten brains found near saloon.\n\n"

response.stream.write "Chickens disappear from farm.\n\n"

response.stream.write "Zombie seen by Walker Ranch!\n\n"

response.stream.close

end

end

EventSource

#controller

class SightingsController < ApplicationController

include ActionController::Live

def alerts

response.headers['Content-Type'] = 'text/event-stream'

response.stream.write "data: Half-eaten brains found near saloon.\n\n"

response.stream.write "data: Chickens disappear from farm.\n\n"

response.stream.write "data: Zombie seen by Walker Ranch!\n\n"

response.stream.close

end

end

function initialize() {

var source = new EventSource('/sightings/alerts');

source.addEventListener('message', function update(event) {

var div = $('<div>').text(event.data);

$('#alerts').append(div);

});

};

$(document).ready(initialize);

<div id="alerts">

</div>

TurboLink Events

function initialize() {

alert('Welcome, Deputy! Check back often for new zombie sightings!');

}

$(document).ready(initialize);

$(document).on('page:load', initialize);

Change a way to add click event

$(document).on('click', '#alert\_resolved', clickAlert);

Show 'loading' when fetching page, under turbolink

<!DOCTYPE html>

<html>

<head>

<title>Zombie Outlaws</title>

<%= stylesheet\_link\_tag "application", media: "all", "data-turbolinks-track" => true %>

<%= javascript\_include\_tag "application", "data-turbolinks-track" => true %>

<%= csrf\_meta\_tags %>

</head>

<body>

<%= yield %>

<div id="loading">Loading...</div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<title>Zombie Outlaws</title>

<%= stylesheet\_link\_tag "application", media: "all", "data-turbolinks-track" => true %>

<%= javascript\_include\_tag "application", "data-turbolinks-track" => true %>

<%= csrf\_meta\_tags %>

</head>

<body>

<%= yield %>

<div id="loading">Loading...</div>

</body>

</html>

$(document).on('page:change', hideLoading);

$(document).on('page:fetch', showLoading);

Disable turbolink e.g.1

<%= link\_to 'Home', welcome\_path, "data-no-turbolink" => true %>

e.g.2

<div id="navigation" <%= link\_to 'Home', welcome\_path, "data-no-turbolink" => true %>>

<div><%= link\_to 'Zombies', zombies\_path %></div>

<div><%= link\_to 'Weapons', weapons\_path %></div>

</div>

Rails Pattern

class UserRegistration

attr\_reader :user

def initialize(params={})

@user = User.new(params)

@user.is\_approved = true if valid\_background\_check?

end

private

# private methods go here

def valid\_background\_check?

!!(@user.valid\_ssn? || @user.valid\_address?)

end

end

NON-AR MODELS User Controller create user by other customized class

class UsersController < ApplicationController

def create

registration = UserRegistration.new(user\_params)

@user = registration.user

if registration.create

redirect\_to @user

else

render :new

end

end

private

def user\_params

params.require(:user).permit(:name, :email, :ssn, :address)

end

end

class UserRegistration

attr\_reader :user

def initialize(params)

@user = User.new(params)

end

def create

if valid\_background\_check?

user.is\_approved = true

end

user.save

end

private

def valid\_background\_check?

!!(@user.valid\_ssn? || @user.valid\_address?)

end

end

Skinny Model

class UserWelcome

attr\_accessor :user

def initialize(user)

@user = user

end

def welcome

send\_welcome\_email

enable\_welcome\_tour

enable\_welcome\_promotion

end

private

def send\_welcome\_email

WelcomeMailer.welcome(@user).deliver

end

def enable\_welcome\_tour

@user.welcome\_tour = true

@user.save

end

def enable\_welcome\_promotion

promo = Promotion.new(name: "Thanks for joining!")

promo.set\_redeemer(@user)

end

end

Remove the welcome function into Userwelcome class

class User < ActiveRecord::Base

def welcome

send\_welcome\_email

enable\_welcome\_tour

enable\_welcome\_promotion

end

private

def send\_welcome\_email

WelcomeMailer.welcome(self).deliver

end

def enable\_welcome\_tour

self.welcome\_tour = true

self.save

end

def enable\_welcome\_promotion

promo = Promotion.new(name: "Thanks for joining!")

promo.set\_redeemer(self)

end

end

Merge Scope

class Item < ActiveRecord::Base

has\_many :reviews

scope :recent, ->{

where('published\_on > ?', 2.days.ago)

.joins(:reviews).merge(Review.approved)

}

end

class Review < ActiveRecord::Base

belongs\_to :item

scope :approved, -> { where(approved: true) }

end

Merge Scope -- remove duplicated part

Review.relevant.merge(Review.pending\_approval)

class Review < ActiveRecord::Base

belongs\_to :item

scope :relevant, -> { where(is\_relevant: true, is\_approved: true) }

scope :pending\_approval, -> { where(is\_approved: false) }

end

Model Concerns

module Reviewable

extend ActiveSupport::Concern

included do

has\_many :reviews, as: :reviewable, dependent: :destroy

end

def reviews\_rating

(reviews.positive.count / reviews.approved.count.to\_f).round(2)

end

end

Classmethods in model concerns

module Reviewable

extend ActiveSupport::Concern

included do

has\_many :reviews, as: :reviewable, dependent: :destroy

end

def reviews\_rating

(reviews.positive.count / reviews.approved.count.to\_f).round(2)

end

module ClassMethods

def with\_no\_reviews

where('id NOT IN (SELECT DISTINCT(reviewable\_id) FROM reviews WHERE reviewable\_type = ?)', self.name)

end

end

end

Decorator

class ItemDecorator

attr\_reader :item

def initialize(item)

@item = item

end

def is\_featured?

@item.ratings > 5

end

def method\_missing(method\_name, \*args, &block)

@item.send(method\_name, \*args, &block)

end

def respond\_to\_missing?(method\_name, include\_private = false)

@item.respond\_to?(method\_name, include\_private) || super

end

end

Decorators collection

class ItemDecorator

def self.build\_collection(items)

items.map { |item| new(item) }

end

end

class ItemsController < ApplicationController

def index

@items = Item.all

@item\_decorators = ItemDecorator.build\_collection(@items)

end

end

**Serialization**

Remove Root Node

class ReviewArraySerializer < ActiveModel::ArraySerializer

self.root = false

end

SENSITIVE PARAMETERS

class Rails4Patterns::Application

# hiding other config for brevity...

config.filter\_parameters += [:password, :security\_answer]

end

Set Ruby Verion

source 'https://rubygems.org'

gem 'rails', '4.0.0'

ruby '2.0.0'

gem 'sqlite3'

gem 'sass-rails', '~> 4.0.0'

gem 'uglifier', '>= 1.3.0'

gem 'coffee-rails', '~> 4.0.0'

gem 'jquery-rails'

gem 'turbolinks'

group :doc do

gem 'sdoc', require: false

end

gem 'puma'

Procfile

web: bundle exec rails s -p $PORT

worker: bundle exec rake worker

urgentworker: bundle exec rake urgent\_worker

scheduler: bundle exec rake scheduler

**Level 6**

1. Side-loaded Object
2. class ItemSerializer < ActiveModel::Serializer
3. attributes :id, :name
4. has\_many :reviews
5. embed :ids, include: true

end

1. On the attributes method, add object.price to data[:price] if a current\_user is present.
2. class ItemSerializer < ActiveModel::Serializer
3. attributes :id, :name
4. def attributes
5. data = super
6. # add conditional here...
7. if current\_user.present?
8. data[:price] = object.price
9. end
10. data
11. end

end

1. Seriaizer Scope
2. class ApplicationController < ActionController::Base
3. serialization\_scope :current\_session

end

**Ruby Bites**

1. Pass a lambda to a method as a block
2. library = Library.new(GAMES)
3. print\_games = lambda { |game| puts "#{game.name} (#{game.system}) - #{game.year}" }

library.each(&print\_games)

1. Capture the block as a proc
2. class Library
3. attr\_accessor :games
4. def initialize(games)
5. @games = games
6. end
7. def each(&block)
8. games.each do |game|
9. block.call game
10. end
11. end

end

1. Optional Block
2. class Library
3. attr\_accessor :games
4. def initialize(games)
5. @games = games
6. end
7. def list
8. games.each do |game|
9. if block\_given?
10. puts yield game
11. else
12. puts game.name
13. end
14. end
15. end

end

* [Status](https://status.github.com/)
* [API](https://developer.github.com)
* [Training](https://training.github.com)
* [Shop](https://shop.github.com)
* [Blog](https://github.com/blog)
* [About](https://github.com/about)
* [Pricing](https://github.com/pricing)
* © 2015 GitHub, Inc.
* [Terms](https://github.com/site/terms)
* [Privacy](https://github.com/site/privacy)
* [Security](https://github.com/security)
* [Contact](https://github.com/contact)
* [Help](https://help.github.com)