BEYOND CRUD

QUICK SETUP

BOOTSTRAP

```
# In the terminal
yarn add bootstrap
```

```
/* application.scss (change file extension to .scss) */
@import "bootstrap/scss/bootstrap"; // picks it up in node_m
```

```
<!-- application.html.erb -->
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial
</head>
```

SIMPLE FORM

```
gem 'simple_form'
```

Then, run:

```
bundle install
rails generate simple_form:install --bootstrap
```

FIRST COMMIT

```
git add .
git commit -m "Rails new with frontend and form gems"
```

CRUD

(Yesterday)

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants
end
```

```
         Prefix  Verb    URI Pattern               Controller#A
    restaurants  GET     /restaurants              restaurants#
                 POST    /restaurants              restaurants#
 new_restaurant  GET     /restaurants/new          restaurants#
edit_restaurant  GET     /restaurants/:id/edit     restaurants#
     restaurant  GET     /restaurants/:id          restaurants#
                 PATCH   /restaurants/:id          restaurants#
                 DELETE  /restaurants/:id          restaurants#
```

SCAFFOLD GENERATOR

- Useful for **quick demo** but not for real projects
- It generates useless files (`scaffolds.scss`, etc.)
- You don't always need all of the 7 CRUD actions

```
rails g scaffold Restaurant name address description:text sta
```

```
rails db:migrate
```

```ruby
# db/seeds.rb
puts 'Cleaning database...'
Restaurant.destroy_all

puts 'Creating restaurants...'
restaurants_attributes = [
  {
    name:         'Dishoom',
    address:      '7 Boundary St, London E2 7JE',
    description:  'Buzzy destination for Indian street food
    stars:        5
  },
  {
    name:         'Pizza East',
    address:      '56A Shoreditch High St, London E1 6PQ',
    description:  'Pizzeria with industrial looks, serving r
    stars:        4
  }
]
Restaurant.create!(restaurants_attributes)
puts 'Finished!'
```

```
rails db:seed
```

BEYOND CRUD

You are not limited to the **seven** routes that RESTful routing creates by default.

Say you want a route to list **five stars** restaurants. Something like:

```
GET /restaurants/top
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    collection do                        # collection => no r
      get 'top'                          # RestaurantsControl
    end
  end
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def top
    @restaurants = Restaurant.where(stars: 5)
  end
end
```

Now we want a route to **display info about the chef of a restaurant**.

```
GET /restaurants/42/chef
```

ADD A CHEF TO RESTAURANTS TABLE

```
rails generate migration AddChefToRestaurants chef:string
rails db:migrate
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    member do                                 # member => restau
      get 'chef'                              # RestaurantsContr
    end
  end
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  before_action :find_restaurant, only: [ :chef ]

  def chef
    @chef_name = @restaurant.chef
  end

  private

  def find_restaurant
    @restaurant = Restaurant.find(params[:id])
  end
end
```

NESTED RESOURCES

WHAT IF?

We also want to store restaurant **reviews**...

- Do we want a review with no restaurant attached to it?
- How can we make the new review form aware of a restaurant?
- Can we use the routes to make our life easier?

What about these routes to **add a review on a restaurant**

- `GET /restaurants/42/reviews/new`
- `POST /restaurants/42/reviews`

What does 42 represent?

MODELS

```
rails generate model Review content:text restaurant:referenc
rails db:migrate
```

```ruby
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
  has_many :reviews, dependent: :destroy
end
```

```ruby
# app/models/review.rb
class Review < ApplicationRecord
  belongs_to :restaurant
end
```

ROUTING

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    resources :reviews, only: [ :new, :create ]
  end
end
```

Will add 2 new routes:

```
               Prefix  Verb  URI Pattern
new_restaurant_review  GET   /restaurants/:restaurant_id/rev
   restaurant_reviews  POST  /restaurants/:restaurant_id/rev
```

```ruby
# app/controllers/reviews_controller.rb
class ReviewsController < ApplicationController
  def new
    # we need @restaurant in our `simple_form_for`
    @restaurant = Restaurant.find(params[:restaurant_id])
    @review = Review.new
  end

  def create
    @review = Review.new(review_params)
    # we need `restaurant_id` to associate review with corre
    @restaurant = Restaurant.find(params[:restaurant_id])
    @review.restaurant = @restaurant
    @review.save
    redirect_to restaurant_path(@restaurant)
  end

  private

  def review_params
    params.require(:review).permit(:content)
  end
end
```

VIEW

How to use `simple_form_for` with nested resources

```erb
# app/views/reviews/new.html.erb
<%= simple_form_for [@restaurant, @review] do |f| %>
  <%= f.input :content %>
  <%= f.submit "add a review", class: "btn btn-primary" %>
<% end %>
```

Helper	**HTTP request**
`form_for(@review)`	POST **`reviews_path`**
`form_for([@restaurant, @review])`	POST **`restaurant_reviews_path`**

QUESTION

Do we **have to** nest routes **because** we have a 1:N relation between 2 models?

🤔

ANSWER

No we **don't**!!!

It's just a convenient way to have **restaurant's id** in **params**!

GUESS THE APPROPRIATE ROUTING

Get one restaurant's reviews

GUESS THE APPROPRIATE ROUTING

We need corresponding `restaurant_id`!

```ruby
Rails.application.routes.draw do
  get "restaurants/:restaurant_id/reviews", to: "reviews#ind
end
```

Or, even simpler, display reviews in restaurant's show:

```ruby
Rails.application.routes.draw do
  get "restaurants/:id", to: "restaurants#show"
end
```

GUESS THE APPROPRIATE ROUTING

Get one review's details

GUESS THE APPROPRIATE ROUTING

We **just** need corresponding `review_id`!

```ruby
Rails.application.routes.draw do
  get "reviews/:id", to: "reviews#show"
end
```

GUESS THE APPROPRIATE ROUTING

Get one review's edit form page

GUESS THE APPROPRIATE ROUTING

Again, we **just** need corresponding `review_id`!

```ruby
Rails.application.routes.draw do
  get "reviews/:id/edit", to: "reviews#edit"
end
```

etc...

SHALLOW NESTING

No need to nest **every** reviews CRUD route in restaurants!

Ask yourself what do you **need** in **params** and you'll know if nesting is necessary.

```
Rails.application.routes.draw do
  get "restaurants/:restaurant_id/reviews",     to: "reviews
  get "restaurants/:restaurant_id/reviews/new", to: "reviews
  post "restaurants/:restaurant_id/reviews",    to: "reviews
  get "reviews/:id",                            to: "reviews
  get "reviews/:id/edit",                       to: "reviews
  patch "reviews/:id",                          to: "reviews
  delete "reviews/:id",                         to: "reviews
end
```

SHALLOW NESTING

Using resources

```
Rails.application.routes.draw do
  resources :restaurants do
    resources :reviews, only: [ :index, :new, :create ]
  end
  resources :reviews, only: [ :show, :edit, :update, :destro
end
```

We don't need restaurant_id for the show, edit, update, destroy or a review!

NAMESPACED ROUTING

WHAT IF?

- You want `restaurants#index` to list **all** restaurants
- You want another `restaurants#index` to list only restaurants **you have created**

ROUTING

```
# config/routes.rb
Rails.application.routes.draw do
  namespace :admin do
    resources :restaurants, only: [:index]
  end
end
```

Will add 1 new route:

```
           Prefix  Verb  URI Pattern          Controller#Acti
admin_restaurants  GET   /admin/restaurants   admin/restauran
```

2 RESTAURANTS CONTROLLER

Usual one

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all
  end
end
```

The new one

```
# app/controllers/admin/restaurants_controller.rb
class Admin::RestaurantsController < ApplicationController
  def index
    # Let's anticipate on next week (with login)
    @restaurants = current_user.restaurants
  end
end
```

VIEW

```erb
<!-- app/views/admin/restaurants/index.html.erb -->
<h1>My list of restaurants</h1>
<% @restaurants.each do |restaurant| %>
  <h2><%= restaurant.name %></h2>
  <p><%= restaurant.address %></p>
<% end %>
```

ALL ABOUT ROUTING

http://guides.rubyonrails.org/routing.html

VALIDATION

```ruby
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
  validates :name, uniqueness: true, presence: true
  validates :address, presence: true
  validates :stars, inclusion: { in: [1, 2, 3, 4, 5] }
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    @restaurant.save
    # Unless @restaurant.valid?, #save will return false,
    # and @restaurant is not persisted.
    # TODO: present the form again with error messages.
    redirect_to restaurant_path(@restaurant)
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address, :sta
  end
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    if @restaurant.save
      redirect_to restaurant_path(@restaurant)
    else
      render :new
    end
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address, :sta
  end
end
```

SIMPLE FORM AND VALIDATIONS

`simple_form` will add validation errors on inputs automagically

In the view, you can use:

```
@restaurant.errors
```

To display all the errors in one block as well if you want.

```erb
<!-- app/views/restaurants/_form.html.erb -->
<%= simple_form_for(@restaurant) do |f| %>
  <% if @restaurant.errors.any? %>
    <div class="errors-container">
      <ul>
        <% @restaurant.errors.full_messages.each do |message
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <!-- [...] all the fields -->
<% end %>
```

Do not forget to add the same logic to the `update` method in your controller!

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def update
    if @restaurant.update(restaurant_params)
      redirect_to restaurant_path(@restaurant)
    else
      render :edit
    end
  end
end
```

READ MORE

Displaying validation errors in views

Your turn! Let's build a CRUD app with just two models.