**CRUD Lab**

All code that needs to be added or typed will be in this color

***Before we start, I encourage you all not to copy and paste anything! Actually understand what’s happening when you’re writing the code, otherwise ask questions. That’s why we’re here. :) Also, I encourage you to go read all the links I have specified. Everything will make a lot more sense. Good luck!***



What is CRUD: Acronym for Create, Read, Update, Destroy

* The acronym CRUD refers to all of the major functions that are implemented in relational database applications
* For any type of model (e.g. users, pokemon, books), you’ll probably need to create, read, update, and destroy them

**Let’s get started!!**

The app: We’re going to create a pokemon app that will implement CRUD functionality for our Pokemon model.

Create a new app in your terminal and then cd into it

rails new pokemon\_app

cd pokemon\_app

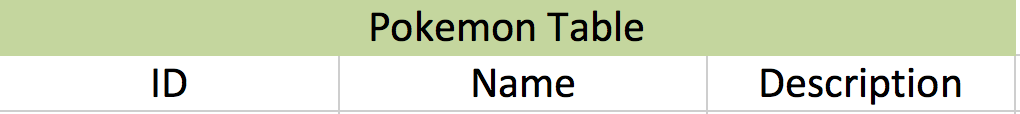
Generate a Pokemon model. A pokemon will have a name (type String) and a description (type Text).

rails generate model Pokemon name:string description:text

Note: It is convention that all models are singular (not plural)

Running the command above should have created a migration file in your db/migrate directory (you can take a look at the file if you want). Remember all it did was add a migration file! It has not actually “migrated” the model to the database. That’s what rake db:migrate is for.

Run rake db:migrate. This adds this table to our database like so:



You now have a pokemon model! :)

Now let’s generate a controller and specify CRUD actions.

rails generate controller Pokemons new create show edit update destroy

Woah that created a lot of files! If you look in your terminal that created the app/views/pokemons directory and added all the views from the actions we specified. It also created a pokemons\_controller.rb and added all the actions as functions. And it appended routes to our routes.rb. The other stuff it added are not important right now.

Note (Rails convention): While models are singular we make controllers plural. So “Pokemons” instead of “Pokemon”

Open routes.rb

You see how it appended the routes:

get 'pokemons/new'

get 'pokemons/create'

get 'pokemons/show'

get 'pokemons/edit'

get 'pokemons/update'

get 'pokemons/destroy'

Let’s delete these routes since we want to create CRUD routes. Now add

resources :pokemons

If we run rake routes to see what routes we have specified you will now see this.

Prefix Verb URI Pattern Controller#Action

pokemons GET /pokemons(.:format) pokemons#index

POST /pokemons(.:format) pokemons#create

new\_pokemon GET /pokemons/new(.:format) pokemons#new

edit\_pokemon GET /pokemons/:id/edit(.:format) pokemons#edit

pokemon GET /pokemons/:id(.:format) pokemons#show

PATCH /pokemons/:id(.:format) pokemons#update

PUT /pokemons/:id(.:format) pokemons#update

DELETE /pokemons/:id(.:format) pokemons#destroy

Remember resources is just a method that creates RESTful routes for you.

What is RESTful routes? : <http://stackoverflow.com/questions/2441962/what-is-restful-routing>

resources :pokemons is literally the same as if you just wrote out these routes out explicitly:

get '/pokemons', to: 'pokemons#index', as: "pokemons"

post '/pokemons', to: 'pokemons#create'

get '/pokemons/new', to: 'pokemons#new', as: 'new\_pokemon'

get 'pokemons/:id/edit', to: 'pokemons#edit', as: 'edit\_pokemon'

get 'pokemons/:id', to: 'pokemons#show', as: 'pokemon'

patch 'pokemons/:id', to: 'pokemons#update'

put 'pokemons/:id', to: 'pokemons#update'

delete 'pokemons/:id', to: 'pokemons#destroy'

Try rake routes and see for yourself. It will be the same thing

Okay, now lets create a form in our new.html.erb in order to create pokemon

<h1> Our Form to Create a Pokemon </h1>

<%= form\_for Pokemon.new, url: {action: "create"} do |f| %>

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

Description: <%= f.text\_area :description %>

<%= f.submit "Create" %>

<% end %>

form\_for is a rails helper thats creates an html form. With the code above, we are creating a form that will enable the browser to send a request to our rails app to create and save a new pokemon.

Now go to <http://localhost:3000/pokemons/new> to check out our form!

Let’s understand how this is actually happening in the context of HTTP and rails in 4 steps:

Don’t get HTTP? A good explanation of http: <http://www.jmarshall.com/easy/http/>

1. When you to go to <http://localhost:3000/pokemons/new> your browser sends a GET request to your server (localhost) running on port 3000
2. When your Rails application receives an incoming HTTP request, the routing engine within Rails is the piece of code that dispatches the request to the appropriate spot in your application.Just think of this as your routes file for now.
3. If you ran rake routes you would see the new action route

Prefix Verb URI Pattern Controller#Action

new\_pokemon GET /pokemons/new(.:format) pokemons#new

which is basically saying on a GET request to pokemons/new go to the pokemons controller new action.

1. So this is when the function in our pokemons\_controller.rb

def new

end

is ran. But how does it know to render the new.html.erb page? There’s no code for that in the new function?There’s actually no code at all! (*P.S. When I say ‘render’ this basically means the controller is telling your rails server to send a response back with the appropriate HTML which then get renders into your browser.)* This is well Rails Magic comes in.

You see this in your controller?

class PokemonsController < ApplicationController

Your PokemonsController inherits from Application Controller which gives you pokemon controller this default functionality. If you want you can do this in you new action:

def new

render ‘new’ # telling the action to render new.html.erb

end

This is basically the same as not explicitly specifying the render function. You can delete after trying it out.

If we refresh the page, we should now see the form (**think about the 4 steps that are happening above when you refresh)**! Now look back at our form\_for in new.html.erb. That’s some wacky Rails Magic huh? How does it know to generate a form that submits a post request to the create url? **Let’s understand this as well in color-coded fashion.**

<%= form\_for Pokemon.new, url: {action: "create"} do |f| %>

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

Description: <%= f.text\_area :description %>

<%= f.submit "Create" %>

<% end %>

This form form creates this HTML in the browser. They are the same thing. form\_for just makes this much more convenient!:

<form accept-charset="UTF-8" action="/pokemons" class="new\_pokemon" id="new\_pokemon" method="post">

<div style="display:none">

<input name="utf8" type="hidden" value="✓">

<input name="authenticity\_token" type="hidden" value="MBww7piL5r9Fs6HJ8nKGj3P5uYYWvlt5H4RL6DxkVW0=">

</div>

Title: <input id="pokemon\_name" name="pokemon[name]" type="text">

Description:

<textarea id="pokemon\_description" name="pokemon[description]"></textarea>

<input name="commit" type="submit" value="Create">

</form>

Note: To see the actual html generated go to the page in your browser, right click, and inspect element. The developer tools should pop up.

url: {action: "create"} is specifying which url to goto namely the create action. But how does it know that this create action is the one in pokemons\_controller.rb? And how does it know to use a POST request? This is where Pokemon.new comes in. Rails knows that since we specified Pokemon.new, this implies that this form is going to be used to *create a new Pokemon.* Therefore, it knows to use a POST request and to use the create action in the pokemon controller which corresponds to the url action="/pokemons" .

Note that actual code Pokemon.new is *only* used so that the form\_for helper knows to generate HTML that creates a form that allows us to send POST requests to create new Pokemon. Pokemon.new **does not actually save anything to the database or do anything else.** That’s what the actual form is for.

P.S.: You might notice some hidden input fields in the HTML. It’s another important security concept in Rails. Check it out here. <http://stackoverflow.com/questions/941594/understand-rails-authenticity-token>

**If you don’t get this explanation ask for help! :)**

**Also there is great explanations about forms here:** <http://guides.rubyonrails.org/form_helpers.html>

Understanding HTTP Verbs

Wait…. so why are we telling the browser to issue a POST request to create Pokemon instead of a GET request? The browser was issuing a GET request earlier to render the new.html.erb view. Or how about PATCH, PUT, and DELETE that was shown when we ran rake routes? These are called HTTP Verbs.

The basic usage for HTTP Verbs in Rails is this

GET: No change to the database. Just for rendering static assets. (e.g. requesting the html form)

PATCH: For updating attribute/s in a instance  
PUT: For completely replacement of an instance

POST: Use to create an instance

DELETE: Use to destroy an instance/s of a model

Additional info

Get vs Post:

<http://blog.teamtreehouse.com/the-definitive-guide-to-get-vs-post>

Patch vs Put: <http://weblog.rubyonrails.org/2012/2/26/edge-rails-patch-is-the-new-primary-http-method-for-updates/>

Google for more!!

Now that we understand HTTP verbs, let’s go back to <http://localhost:3000/pokemons/new> where are form is. You might be wondering why I color coded name="pokemon[name]" and

name="pokemon[description]" . We’ll get to that in a moment.

Try typing stuff in the form and submitting it! YAY! Did it work? You we’ll see that it redirected you to the create.html.erb form. NO ERRORS so it must have worked?? Let’s find out….

Go to your rails console by typing rails console in your terminal and query all the Pokemon.all

You will see #<ActiveRecord::Relation []> which means there is no Pokemon in our database. It didn’t work :(.

Moreover, **it’s important that you check what your rails server is logging out from time to time. There’s useful information there.** Let’s take a look at what is said. Go to you terminal where the rails server is running.

Started POST "/pokemons" for 127.0.0.1 at 2014-10-01 12:40:35 -0700

Processing by PokemonsController#create as HTML

Parameters: {"utf8"=>"✓", "authenticity\_token"=>"MBww7piL5r9Fs6HJ8nKGj3P5uYYWvlt5H4RL6DxkVW0=", "pokemon"=>{"name"=>"asdf", "description"=>"asdfasdf"}, "commit"=>"Create"}

Rendered pokemons/create.html.erb within layouts/application (0.1ms)

Completed 200 OK in 70ms (Views: 68.4ms | ActiveRecord: 0.0ms)

So the browser made the post request correctly which was received by the server. And then our rails app ran the PokemonsController#create function. And then it rendered the create.html.erb. But the server is not saying that a new Pokemon is created. Why? If we go to our pokemon\_controller.rb, we see are create function defined.

def create

end

Well of course is not making a new Pokemon because we didn’t specify that in our create function. Aren’t we dummies :). And it’s just rendering the create.html.erb because that the default functionality because of the inheritance of Application Controller I talked about before. So let’s add some code.

def create

@pokemon = Pokemon.new(params[:pokemon]) #create a new pokemon

@pokemon.save #save the pokemon

redirect\_to pokemon\_path(@pokemon) # redirect to the show action

end

So what is params[:pokemon]? Remember name="pokemon[name]" and

name="pokemon[description]" in our form (see the HTML above)? This tells the browser to send the values we write in our form in the POST request. Our rails server receives these parameters and stores it into a ruby hash called params. Your rails server also tells you what the parameters it’s receiving. So if we submit a form with

name: Derrick

description: The pokemon that lurks in the swamps

Our rails server will log the params.

Parameters: {"utf8"=>"✓", "authenticity\_token"=>"MBww7piL5r9Fs6HJ8nKGj3P5uYYWvlt5H4RL6DxkVW0=", "pokemon"=>{"name"=>"Derrick", "description"=>"The pokemon that lurks in the swamps"}, "commit"=>"Create"}

So basically

params = {"utf8"=>"✓", "authenticity\_token"=>"MBww7piL5r9Fs6HJ8nKGj3P5uYYWvlt5H4RL6DxkVW0=", "pokemon"=>{"name"=>"Derrick", "description"=>"The pokemon that lurks in the swamps"}, "commit"=>"Create"}

params[:pokemon] = {"name"=>"Derrick", "description"=>"The pokemon that lurks in the swamps"}

Therefore, @pokemon = Pokemon.new(params[:pokemon]) is the same as

@pokemon = Pokemon.new( {"name"=>”Derrick", "description"=>"The pokemon that lurks in the swamps"})

See more about params here. Section 7.3 gives more examples. <http://guides.rubyonrails.org/form_helpers.html#understanding-parameter-naming-conventions>

Go ahead and try to submit the form with some values. Uh oh. We’re getting a ActiveModel::ForbiddenAttributesError

**Strong parameters (security)**

I’m not going to go into detail about strong parameters here. But it’s EXTREMELY important which is why rails errors out. The basic idea is that a hacker can use CURL or some other tool like Postman (<http://bit.ly/18JpMha> ) to issue requests to change parameters we don’t want them to change. For example, say we had a User model that had a boolean admin attribute. If it is true, then that user could access admin functionality. A hacker may make a request with one of the values being admin:true which changes the admin attribute to true for a particular user. We don’t want that.

Learn more about Strong Parameters here: <http://code.tutsplus.com/tutorials/mass-assignment-rails-and-you--net-31695>

So let’s add this code in the **bottom** of a pokemons\_controller.rb

private

# filters out unwanted params

def pokemon\_params

params.require(:pokemon).permit(:name, :description)

end

pokemon\_params is a function the requires the pokemon key in the params and filters out the params in which only name and description can be changed.

The private on top is just good practice. Were making this function private so that it can only be used in this controller. In ruby anything under the private declaration is considered a private function so make sure *only* pokemon\_params function is under there.

Now change params[:pokemon] to pokemon\_params. Now only appropriate attributes can be added or changed in a Pokemon instance.

def create

@pokemon = Pokemon.new(pokemon\_params) #create a new pokemon

@pokemon.save # save the pokemon

redirect\_to pokemon\_path(@pokemon) # redirect to the show action

end

Okay now that we get the 1st line, the 2nd line in the function is pretty straightforward. This is where we actually save the pokemon to the database.

**Note:** All the 1st and 2nd line is doing can be done in the rails console! It’s literally the same thing except in a controller function. For example, you can create a Pokemon in the rails console by doing

> @sam\_pokemon = Pokemon.new({"name"=>"Sam", "description"=>"The invincible one"})

> @sam\_pokemon.save

But the last line can be a little confusing: redirect\_to pokemon\_path(@pokemon)

If we run rake\_routes again we see the show route like so. Notice the Prefix column:

Prefix Verb URI Pattern Controller#Action

pokemon GET /pokemons/:id(.:format) pokemons#show

If a Prefix exists for a route, all this does is create a corresponding method that returns to the url for the action. All Prefixes are followed by the Suffix \_path or \_url. Both are interchangeable.

So pokemon\_path or pokemon\_url is just a method for the url corresponding to the show action. If we run rake routes we see that the resources method also specifies some prefixes for other routes.

Prefix Verb URI Pattern Controller#Action

pokemons GET /pokemons(.:format) pokemons#index

new\_pokemon GET /pokemons/new(.:format) pokemons#new

edit\_pokemon GET /pokemons/:id/edit(.:format) pokemons#edit

pokemon GET /pokemons/:id(.:format) pokemons#show

Therefore:

pokemons\_path returns url for index action

new\_pokemon\_path returns url for new action

edit\_pokemon returns url for edit action

pokemon\_path returns url for show action

So when we see

redirect\_to pokemon\_path(@pokemon)

this is saying we want to redirect\_to the show action. We need to pass in the @pokemon because the show action has to know what pokemon to show. For example, if the pokemon you have just created and saved had id:5 then

pokemon\_path(@pokemon) will redirect to ‘/pokemons/5’

We can even make this more rails like by just doing

def create

@pokemon = Pokemon.new(pokemon\_params)

@pokemon.save

redirect\_to @pokemon

end

With redirect\_to @pokemon rails inherently knows to go to the show action. This is because were redirecting to a specific pokemon. Makes sense right?

Okay… that’s a lot of stuff so let’s get hyped up again with you know who before going on!



Now that we got the juices pumping, go ahead and go back to our form to create pokemon <http://localhost:3000/pokemons/new>

Enter some values and submit the form! You should see the show view rendered and if you go to your console and type Pokemon.all you should see your newly created Pokemon!! Moreover, you can go to your rails server that’s logging stuff and see that it says it created a Pokemon.

Parameters: {"utf8"=>"✓", "authenticity\_token"=>"MBww7piL5r9Fs6HJ8nKGj3P5uYYWvlt5H4RL6DxkVW0=", "pokemon"=>{"name"=>"Pikachu", "description"=>"awesome!!"}, "commit"=>"Create"}

(0.2ms) begin transaction

SQL (0.7ms) INSERT INTO "pokemons" ("created\_at", "description", "name", "updated\_at") VALUES (?, ?, ?, ?) [["created\_at", "2014-10-02 00:58:54.420804"], ["description", "awesome!!"], ["name", "Pikachu"], ["updated\_at", "2014-10-02 00:58:54.420804"]]

(0.8ms) commit transaction

Redirected to http://localhost:3000/pokemons/5

Awesome! We just finished the C in CRUD. But the show view is pretty bland. The purpose of it is to actually visually show the information of the pokemon. So let’s do some editing to the controller and view.

# In pokemon\_controller.rb

def show

@pokemon = Pokemon.find(params[:id])

end

<!-- In your show.html.erb -->

<h1>Showing Pokemon Details</h1>

<h3>Name</h3>

<%= @pokemon.name %>

<h3>Description</h3>

<%= @pokemon.description %>

Refresh and WALA! :) Just like that we just completed the R in CRUD by querying and reading a Pokemon from the database. I’ll quickly go over what’s happening

1. You probably have a different id for your pokemon, but for this example let’s say my id for the pokemon is 2. When we refreshed the page of <http://localhost:3000/pokemons/2> your browser sends a get request with this URL.
2. If we look at the parameters in our rails server we see

Parameters: {"id"=>"2"}

Rails knows to make params = {"id"=>"2"} because of the structure of our URL.

1. Rails then directs the request to the show action. Remember we have the params variable available and we use this to find the Pokemon.

@pokemon = Pokemon.find(params[:id]) # where params[:id] = 2

**A challenge question is why don’t we need to implement strong parameters here? I’ll post the answer in piazza later.**

1. So now we can use @pokemon in our view and you know the rest!!

For kicks let’s add a link to our show view that will take us back to the new form.

<h1>Showing Pokemon Details</h1>

<h3>Name</h3>

<%= @pokemon.name %>

<h3>Description</h3>

<%= @pokemon.description %>

</br>

<%= link\_to 'Go back to form to create pokemon', new\_pokemon\_path %>

Remember new\_pokemon\_path is just a variable for the url for the new action. We know this by the prefix when we ran rake routes. link\_to is a rails view helper that generates anchor tag html (which are for links).

The corresponding html generated was

<a href="/pokemons/new">Go back to form to create pokemon</a>

Refresh the page and you should now see the link in your show view. Go ahead and click on it and it’ll take you back to your form to create another pokemon.



Yeah that lab was long, but my intentions was that you actually understand what’s happening other than just having Rails do it’s magic. But really though, you should be as pumped as this baby right now!!

If this is your first time ever creating a user int