Rapid Application Development

COSC2675 2017 Week 6

Dr. Andy Song
andy.song@rmit.edu.au

Last Week

- Rails introduction and philosophy
- Technical overview
- The hello application
- The mechanism / directory structure of Rails
- Adding style sheets
- Understand layouts **
- Adding images
- The `layout` method
- Template working with a layout

Action Pack

- Last week we introduced routing, controllers and views. These are the functionalities of Action Pack, a core component of Rails. The documentation is here:
  `http://api.rubyonrails.org/files/actionpack/README_rdoc.html`
- Views – generating HTML (via ERB templates/layouts)
- Controllers – the logical center.
  - Coordinating the interaction between users, views and "databases"
  - Routing external requests to internal actions, people-friendly URLs
  - Managing caching
  - Managing helpers
  - Managing sessions

Models

- Controllers are important, but don't get too caught up in them. They are supposed to be skinny.
- Model is the M of MVC, the foundation of an application.
- Models are where all the persistent data is managed.
- Models can connect to databases through ORM (object-relational mapping).
- Rails' ORM is Active Record
- Let us start from something simple, form

A guestbook using form

- Create a new application:

```
$ rails new guestbook
$ cd guestbook
$ rails generate controller entries sign_in
```

- Now we have a `sign_in` method in

  `app/controllers/entries_controller.rb`

- What will you see in browser, say `https://….c9users.io/`
- How about `https://….c9users.io/entries` or

  `https://….c9users.io/entries/sign_in`
- We haven't specified the route for incoming requests yet.

'Set' the guestbook application

- **Update the route:** `config/route.rb`

```
get 'entries/sign_in'
get  'entries/sign_in' => 'entries#sign_in'
post 'entries/sign_in' => 'entries#sign_in'
```

- **Update the view:** `app/views/entries/sign_in.html.erb`

```
<h1> Hello <%= @name %> </h1>
<%= form_tag action: 'sign_in' do %>
  <p> Enter your name:
  <%= text_field_tag 'visitor_name', @name %> </p>
  <%= submit_tag 'Sign in' %>
<% end %>
```

- **Update the controller:** `app/controller/entries_controller.erb`

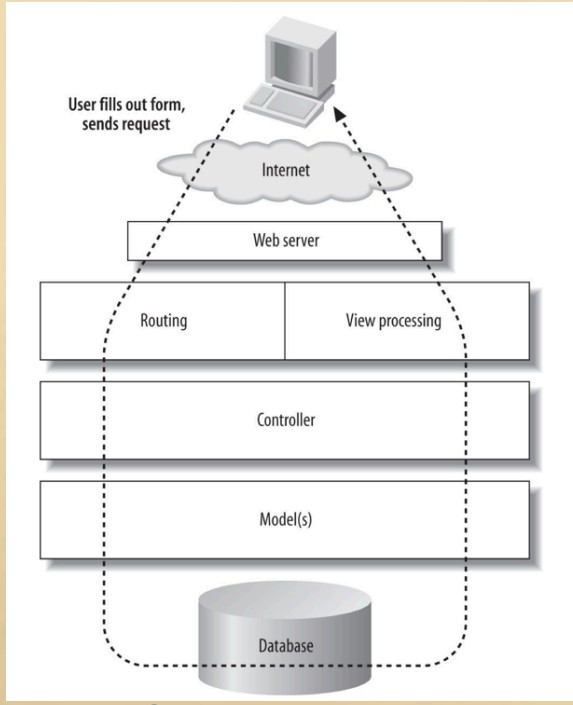
```
class EntriesController < ApplicationController
  def sign_in
    @name = params[:visitor_name]
  end
end
```

View it in a browser

- **Launch the server by** `$ rails s –p $PORT -b $IP`

- The controller is now receiving data from the user and passing it to a view.

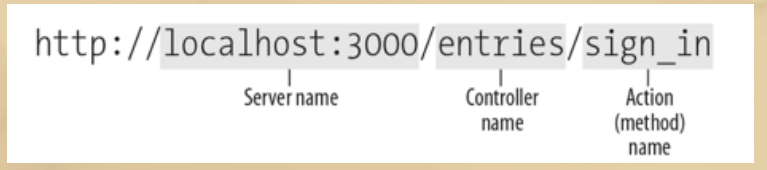
How it works?



(Learning Rails 5, Mark Locklear Eric Gruber)

Rails takes care of a lot of work

- Listen to the request `http://.../entries/sign_in`
- Look at `config/routes.rb` to see the right route
  `get 'entries/sign_in' => 'entries#sign_in'`
- Call `entries_controller` method `sign_in`
- Render the view using files in `app/views` and show the form.
- Get data from the submitted form
  `post 'entries/sign_in' => 'entries#sign_in'`
- Re-display the form page and continue listening to requests



The form generated by Rails

```
<!DOCTYPE html> <html> <head> <title>Guestbook</title>
<meta name="csrf-param" content="authenticity_token" />
<meta name="csrf-token" content="…….=" />
<link rel="stylesheet" media="all" href="…..
……. data-turbolinks-track="reload"></script> </head> <body>

<h1> Hello RAD </h1>
<form action="/entries/sign_in" accept-charset="UTF-8" method="post">
<input name="utf8" type="hidden" value="&#x2713;" />
<input type="hidden" name="authenticity_token" value="/
SpKDCVAQ0oag6j4YEfuY95lWcGWgylBlwl+PmzLG4QKJWYsB5I01tEMz5ohqKl/3AYK
+SifR1o9dXU5O/svEg==" />
<p> Enter your name:
<input type="text" name="visitor_name" id="visitor_name" value="RAD" />
</p>

<input type="submit" name="commit" value="Sign in"
                                   data-disable-with="Sign in" />
</form> </body> </html>
```

Connect to a DB through a model

- Check whether SQLite is install `$ sqlite –version`
- Create a model `$ rails generate model entry`

```
Running via Spring preloader in process 11245
    invoke  active_record
    create    db/migrate/[Time Stamp]_create_entries.rb
    create    app/models/entry.rb
    invoke    test_unit
    create      test/models/entry_test.rb
    create      test/fixtures/entries.yml
```

- Content of `db/migrate/[Time Stamp]_create_entries.rb:`

```
class CreateEntries < ActiveRecord::Migration[5.0]
  def change
    create_table :entries do |t|
      t.timestamps
    end
  end
end
```


The migrate rb

- A table with plural form of the model name is created
- So table books for model book, entries for entry, mice for mouse, waters for water, sheep for sheep, gooses for goose, men for man, wolves for wolf, oxen for ox, children for child.
- Method `change` replaced `self.up` and `self.down` for undo.
- Add one line:

```
class CreateEntries < ActiveRecord::Migration[5.0]
  def change
    create_table :entries do |t|
      t.string :name
      t.timestamps
    end
  end
end
```

- This line is to create a column of string to store names.

Migration

- To make the last change take effect, run `$ rails db:migrate`
- Migrations are a convenient way to change your database schema over time.
- Each migration can be considered as a new version of the DB
- A schema starts off with nothing in it.
- Each migration modifies it to add or remove tables, columns or entries.
- It is managed by ActiveRecord which controls the versions.
- You can
```
$ rails db:migrate:redo STEP=3
$ rails db:rollback STEP=3
$ rails db:migarte VERSION=20170301120000
```


The skinny model

- Open `app/models/entry.rb,` we see simply

```
class Entry < ApplicationRecord
end
```

- Before Rails 5.0, a model is a direct sublass of ActiveRecord.
- Now it is through ApplicationRecord, open `app/models/application_record.rb`

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

- It tries to be an abstract class although Ruby does not have abstract method and class in its syntax.

The controller

- Edit the controller app/controllers/entries_controller.rb

```
class EntriesController < ApplicationController
 def sign_in
   @name = params[:visitor_name]
   @entry = Entry.create({:name => @name})
end
```

- This line effectively does three tasks for each entry

```
@entry = Entry.new
@entry.name = @name    # @name is from the form
@entry.save
```

- Name @entry is not special, can be @data, @info, @foo etc.
- The new version is better as its avoids empty entries and read all entries.

```
unless @name.blank?
  @entry = Entry.create({: name = > @name})
end

@entries = Entry.all
```

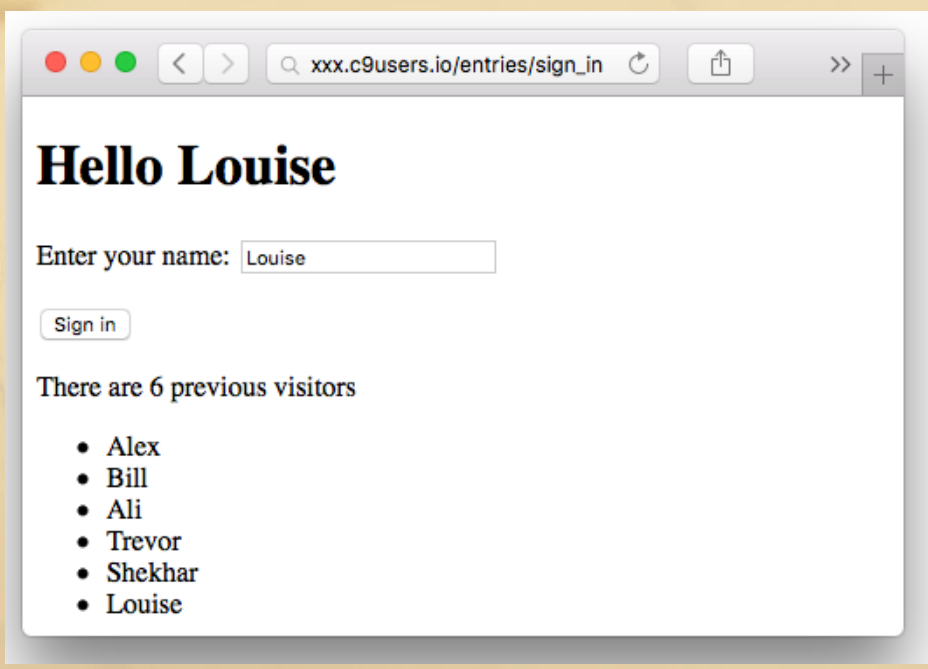

Then the view

- Edit the view: app/views/entries/sign_in.html.erb

```
<h1> Hello <%= @name %> </h1>
<%= form_tag action: 'sign_in' do %>
  <p> Enter your name:
  <%= text_field_tag 'visitor_name', @name %> </p>
  <%= submit_tag 'Sign in' %>
<% end %>


<p> There are <%= @entries.size %> previous visitors </p>
<ul>
<% @entries.each do |entry| %>
   <li> <%= entry.name %> </li>
<% end %>
</ul>
```

Now you should see



Take a look at the console

```
Started POST "/entries/sign_in" for [IP] at [TimeStamp]
Processing by EntriesController#sign_in as HTML
  Parameters: {"utf8"=>"✓", "authenticity_token"=>"pw..[cut]...==",
  "visitor_name"=>"Louise", "commit"=>"Sign in"}
  (0.1ms)  begin transaction
  SQL (1.1ms)  INSERT INTO "entries" ("name", "created_at", "updated_at")
  VALUES (?, ?, ?)  [["name", "Louise"], ["created_at", [TimeStamp]
  UTC], ["updated_at", [TimeStamp] UTC]]
   (15.0ms)  commit transaction
  Rendering entries/sign_in.html.erb within layouts/application
   (0.3ms)  SELECT COUNT(*) FROM "entries"
  Entry Load (0.2ms)  SELECT "entries".* FROM "entries"
  Rendered entries/sign_in.html.erb within layouts/application (153.6ms)
Completed 200 OK in 196ms (Views: 175.3ms | ActiveRecord: 16.8ms)
```

Rails is working hard behind the scene!

Model Methods

- `Entry.all,` `Entry.find(1).id`
- `Entry.first,` `Entry.last.name`
- `Entry.where(name: “Alex”)`
- `Entry.limit 3, Entry.limt(3).offset(2)`
- `Entry.order(:name)` `# and many DB query methods!`
- The Rails documentation also has some
  `http://guides.rubyonrails.org/active_record_querying.html`
- These methods are from Active Record. See details of these methods on Rails API: `api.rubyonrails.org`
- These methods can also be called in Rails console!

```
>>Entry.find(1)
  Entry Load (0.3ms)  SELECT  "entries".* FROM "entries" WHERE
"entries"."id" = ? LIMIT ?  [["id", 1], ["LIMIT", 1]]
=> #<Entry id: 1, name: "Alex", created_at: [TimeStamp], updated_at:
[TimeStamp]>
```


Scaffolding a model

- Scaffolding is one of the main contributing factors for rapid Rails development.
- Let us try that on the guest book application:

```
$ rails new guestbook_2
$ cd guestbook_2
$ rails generate scaffold Person name:string
$ rails generate controller entries sign_in [no scaffolding]
```

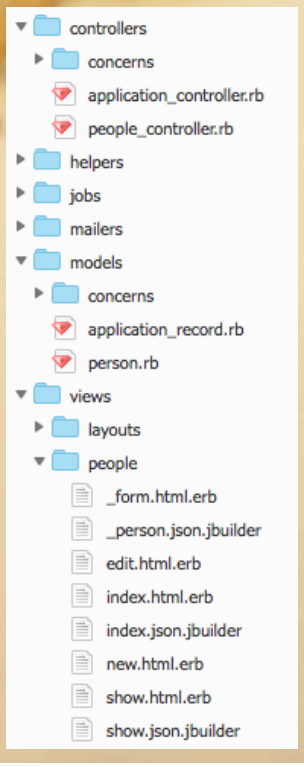
- It generates scaffolding, around model `Person` with one attribute `name`. Note the table name is `people` not `persons.`
- Try to view it on `http://… c9users.io/people`

  What do you see?
- Try again after running `$ rails db:migrate`

What happened in scaffolding

- Scaffolding allows Rails create a list of components
  - a model with tests
  - a data migration to establish the tables for the model
  - a new route to map user requests to the controller
  - a controller to connect different components and pass data
  - four views (index, edit, show and new) and a partial form
  - tests for the controller
  - an empty file for helper methods
  - a CoffeeScript file for scripting the pages
  - Stylesheets for all of those views

What happened in scaffolding



- `config/routes.rb`

```
Rails.application.routes.draw do
  resources :people
end
```

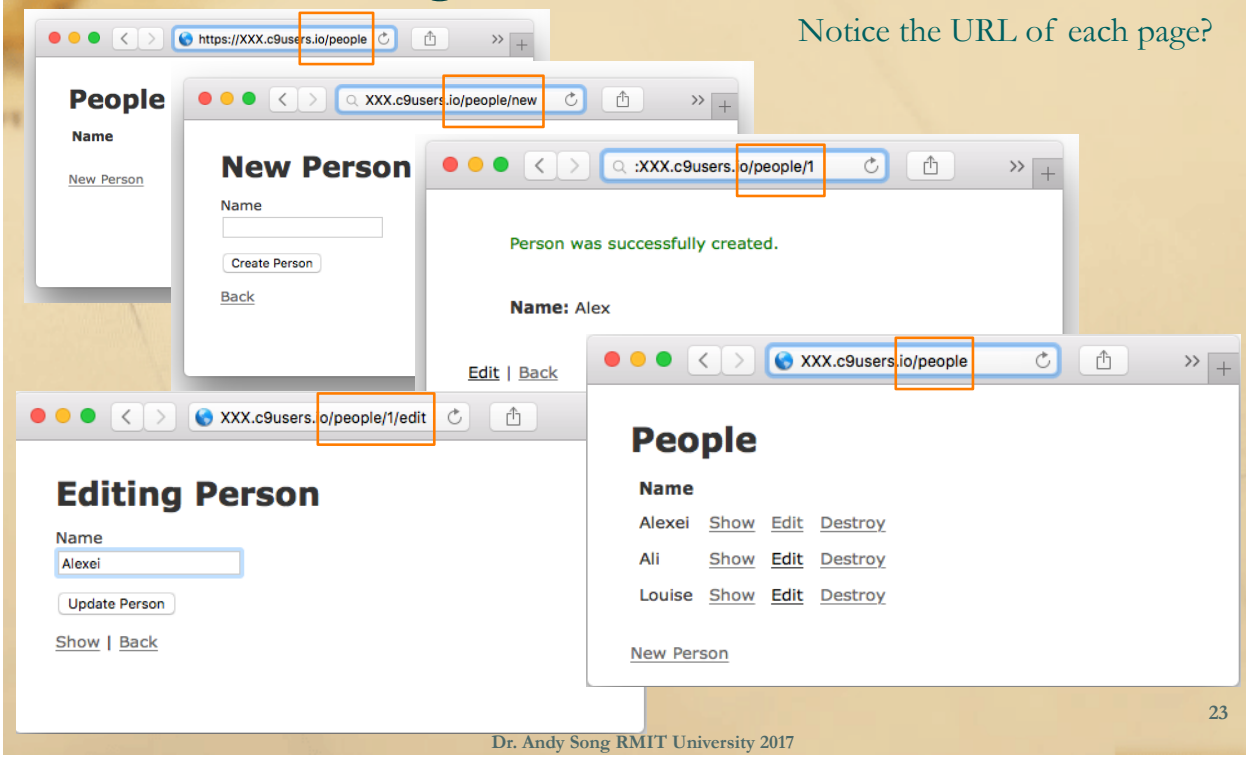
- Not much in model `app/models/person.rb`

```
class Person < ApplicationRecord
end
```

- 70+ lines added automatically to the controller `app/controllers/people_controller.rb`
- The view is a bit complicated.

Interacting with the site

Notice the URL of each page?



RESTful Rails

- The URLs are meaningful:
  `…/people/edit`
  `…/people/new`
  `…/people, …/people/1, …/people/4`
- The actions are visible through these URLs
- You can bookmark a data resource say `…/people/4` or an action `…/people/2/edit`
- Admins can manage web traffic without worrying about disrupting the application.
- A neat fit with Rails' MVC architecture.
- Easier to behave RESTfully.


REST

- REpresentational State Transfer, an architecture defined by Roy Fielding in his 2000 PhD thesis at UC Irvine.
- Allows access text-based web resources using a uniform and pre-defined set of stateless operations.
- Does not create new techniques.
- Based on HTTP request, GET = read, POST = write, DELETE = destroy, PUT = update etc.
- Supports CRUD (create, read, update, destroy) which are common persistent resource operations, often seen in DB, Web, Files.
- Simpler than SOAP (Simple Object Access Protocol, XML based) and WSDL (Web Services Description Language, XML based)
- Other properties: good scalability, performance, visibility, portability, modifiability, reliability.


REST

- Requires a shift in the way developers think about controllers and writing web applications.
- Controllers can be verbs. Each controller implements the same verbs.
- The controller becomes a standardized connection to data model.
- URLs connect the client to a resource (noun) on the server in predictable ways.
- Based on HTTP request, GET = read, POST = write, DELETE = destroy, PUT = update etc.
- It is not only for HTML, but can be used on resources for JSON, Ajax.

RESTful controller

- Requires a shift in the way developers think about controllers and writing web applications.
- Controllers can be verbs. Each controller implements the same verbs.
- The controller becomes a standardized connection to data model.
- URLs connect the client to a resource (noun) on the server in predictable ways.
- Based on HTTP request, GET = read, POST = write, DELETE = destroy, PUT = update etc.
- It is not only for HTML, but can be used on resources for JSON, Ajax.

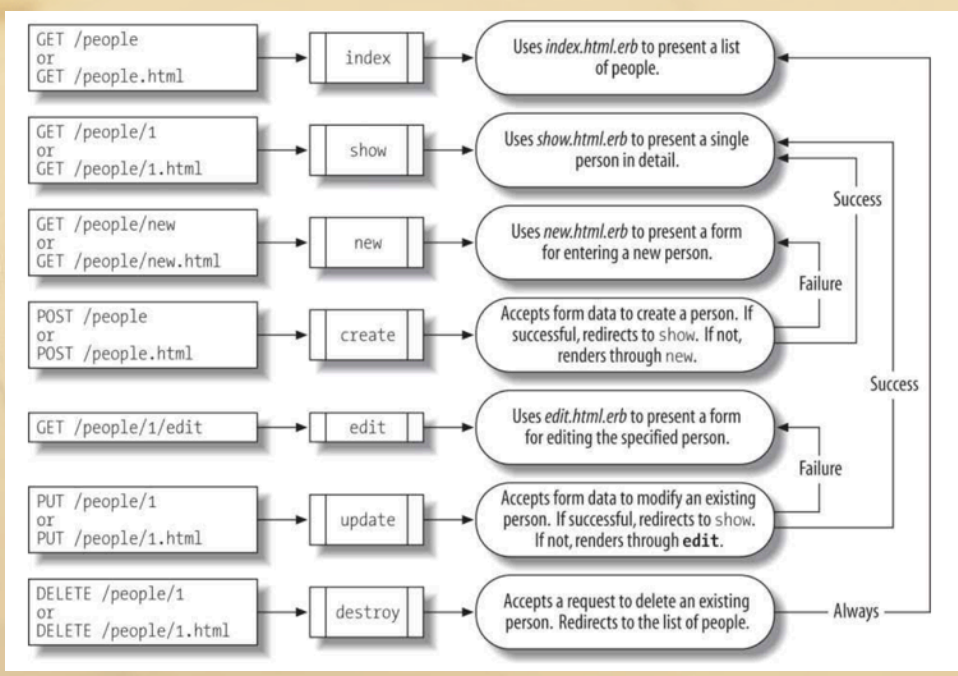
RESTful Routes

- Remember what is in `config/routes`
  `resources :people`
- If we run this command `$ rails routes`

```
      Prefix Verb   URI Pattern                  Controller#Action
      people GET    /people(.:format)            people#index
             POST   /people(.:format)            people#create
  new_person GET    /people/new(.:format)        people#new
 edit_person GET    /people/:id/edit(.:format)   people#edit
      person GET    /people/:id(.:format)        people#show
             PATCH  /people/:id(.:format)        people#update
             PUT    /people/:id(.:format)        people#update
             DELETE /people/:id(.:format)        people#destroy
```

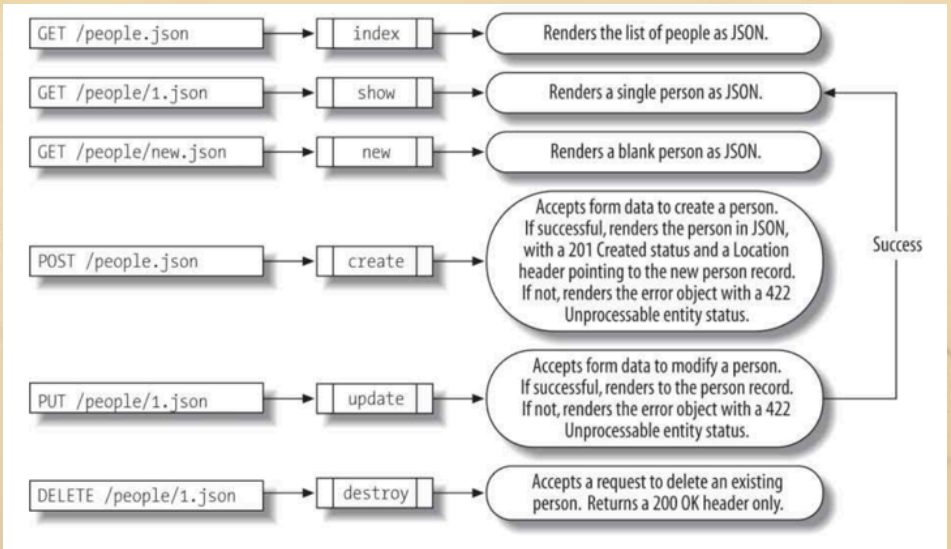
- Resources give a set of RESTful URLs.
- HTTP requests are mapped to actions. Meaning a URL such as `GET /people/1/edit` is processed by `people#edit`

RESTful Routes



(Learning Rails 5, Mark Locklear Eric Gruber)

Can work on JSON files



(Learning Rails 5, Mark Locklear Eric Gruber)

Can work on JSON files

- You can try in a browser `http://…..../people.json`
  If we run this command `$ rails routes`
- The output of JSON files can be viewed at the browser

```
[{"id":1,"name":"Alexei","created_at":"[TimeStamp]",
"updated_at":"[TimeStamp]","url":"https://
[acount].c9users.io/people/1.json"},
{"id":2,"name":"Ali","created_at":"[TimeStamp]",
"updated_at":"[TimeStamp]","url":"https://
[acount].c9users.io/people/2.json"},
{"id":4,"name":"Louise","created_at":"[TimeStamp]",
"updated_at":"[TimeStamp]","url":"https://
[acount].c9users.io/people/4.json"}]
```

- HTTP requests are mapped to actions. Meaning a URL such as `GET /people/1/edit` is processed by `people#edit`

```ruby
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :destroy]

  # GET /people
  # GET /people.json
  def index
    @people = Person.all
  end

  # GET /people/1
  # GET /people/1.json
  def show
  end

  # GET /people/new
  def new
    @person = Person.new
  end

  # GET /people/1/edit
  def edit
  end

  # POST /people
  # POST /people.json
  def create
    @person = Person.new(person_params)

    respond_to do |format|
      if @person.save
        format.html { redirect_to @person, notice: 'Person was successfully created.' }
        format.json { render :show, status: :created, location: @person }
      else
        format.html { render :new }
        format.json { render json: @person.errors, status: :unprocessable_entity }
      end
    end
  end
```

```ruby
39
40   # PATCH/PUT /people/1
41   # PATCH/PUT /people/1.json
42   def update
43     respond_to do |format|
44       if @person.update(person_params)
45         format.html { redirect_to @person, notice: 'Person was successfully updated.' }
46         format.json { render :show, status: :ok, location: @person }
47       else
48         format.html { render :edit }
49         format.json { render json: @person.errors, status: :unprocessable_entity }
50       end
51     end
52   end
53
54   # DELETE /people/1
55   # DELETE /people/1.json
56   def destroy
57     @person.destroy
58     respond_to do |format|
59       format.html { redirect_to people_url, notice: 'Person was successfully destroyed.' }
60       format.json { head :no_content }
61     end
62   end
63
64   private
65     # Use callbacks to share common setup or constraints between actions.
66     def set_person
67       @person = Person.find(params[:id])
68     end
69
70     # Never trust parameters from the scary internet, only allow the white list through.
71     def person_params
72       params.require(:person).permit(:name)
73     end
74 end
```


RESTful Methods

- `before_action :set_person, only: [:show, :edit, :update, :destroy]`
- The private callback `set_person`
- The `new` method does not touch the database, only in memory.
- The controller simply pass the model to the view `new.html.erb`, without worrying about the schema etc.
- `Person.new(person_params)` is another callback
- `respond_to` method is from ActionController, allows responses in various formats on the same data.


Remarks

- **ActiveScaffold** is much more powerful.
- It provides AJAX-ified CRUD interface and supports pagination, sorting, search, CSS, themes. Good for admin interfaces
- **Micro-applications**
- Rails is well equipped for large-scale web applications. However it can be used to quickly build a small application, e.g. address list, glossaries, expense tracking.
- Rails doesn't enforce RESTful resources.
- It is possible to just use GET/POST or the mix.


Summary

- **Action Pack**
- **Model**
- **Controller**
- **DB migration**
- **Working with a simple web form**
- **Scaffolding**
- **REST**
- **RESTful resources (with different formats)**
- **RESTful methods**