Intro to Rails

Aaron Bartell   abartell@krengeltech.com

LITMIS.COM - Open Source for i

"Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days."

-Tim O'Reilly, Founder of O'Reilly Media

*

*… an open-source **web framework** written in the Ruby language that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.*

Programming 5250 green screens with RPG was very efficient. Ruby On Rails (aka Rails) accomplishes the same, but in the modern web world.

- **Est 2004 by David Heinemeier Hansson**, in 2006 Apple ships with OSX 10.5 "Leopard"
- **Separation of concern** (model-view-controller, "helpers", mailers, presenters)
- **Generate application components** with custom gems and configurations.
- **Multiple environments** (i.e. dev, test, prod, <custom>)
- **ActiveRecord** pattern
- [i18n features](#) (**internationalization**) by default when using `rails` command.
- Includes Rake, a **task management** tool (i.e. rake db:migrate)
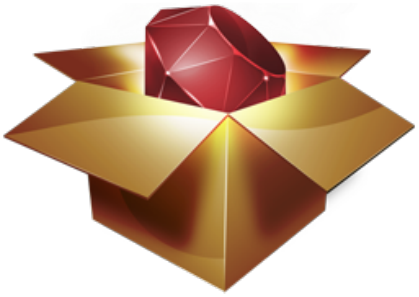- **Unit testing**
- **Asset pipeline**

Really, too many features and benefits to list so let's dive in!

RubyOnRails.org – Framework home
Guides.RubyOnRails.org/getting_started.html – Quick learning, formal documentation
RailsCasts.com - Excellent video resource for learning about all things Rails
TeamTreehouse.com - Comprehensive Ruby/Rails/HTML/CSS/Javascript training

RubyGems… *simplify the process of installing, removing, updating and managing Ruby libraries and their dependencies.*

Rails is a collection of gems

- ActionMailer - Easy email delivery and testing

- ActionPack - Parse request, routing, and controller implementation.

- ActionView - View template lookup, rendering, view "helpers", and more.

- ActiveModel - Allows ActionPack to interact with non-ActiveRecord models.

- ActiveRecord - Connects classes to relational database tables with zero initial configuration.

- ActiveSupport - Utility classes and standard library extensions that were found useful for the Rails framework

rubygems.org - Home website
linuxjournal.com/article/8967 – RubyGem history

Convention Over Configuration (CoC)

"…means that Rails makes assumptions about what you want to do and how you're going to do it, rather than requiring you tweak every little thing through endless configuration files."

Examples

- View (as in M**V**C) named same as controller method

- Config files still exist, but the most commonly used defaults are used.

- Things like jquery are integrated by default

guides.rubyonrails.org/active_record_basics.html#convention-over-configuration-in-active-record

Naming of ActiveRecord
Models and Tables.

Model / Class	Table / Schema
Post	posts
LineItem	line_items
Deer	deers
Mouse	mice
Person	people

HTTP Verb	Path	Action	Used for
GET	/photos	index	display a list of all photos
GET	/photos/new	new	return an HTML form for creating a new photo
POST	/photos	create	create a new photo
GET	/photos/:id	show	display a specific photo
GET	/photos/:id/edit	edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	update	update a specific photo
DELETE	/photos/:id	destroy	delete a specific photo

New App From Scratch

The RAILSNEW command creates a new Rails application, including bundling necessary gems.

```
RAILSNEW
  VRM(*V200) --version of PowerRuby
  PLACE(*WWW) -- *WWW or *HOME
  RAILSAPP(A2222) -- Name of Rails app. /www/A2222/htdocs/ A2222
  DBUSR(A2222)  -- CRTUSRPRF A2222 with below DBPASS
  DBPASS(A2222)
  DBROOT(A2222) -- Run SQL: CREATE COLLECTION  A2222_D; CREATE COLLECTION
A2222_T
  HTTPSRV(A2222) -- /www/A2222
  EXTPORT(2222) -- /www/A2222/conf/httpd.conf  Listen *:2222
  INTPORT(2202) -- thin start -p 2202
```

Command `bundle install --local` is run so all gems are resolved and the Gemfile.lock file is created.

It also runs the following commands to start Apache and Thin

```
STRTCPSVR HTTPSVR(A2222)  SERVER(*HTTP)
RAILSSVR ACTION(*START)
        APP('/www/A2222/htdocs/A2222')
        VRM(*V200)
        PORT(2202)
```
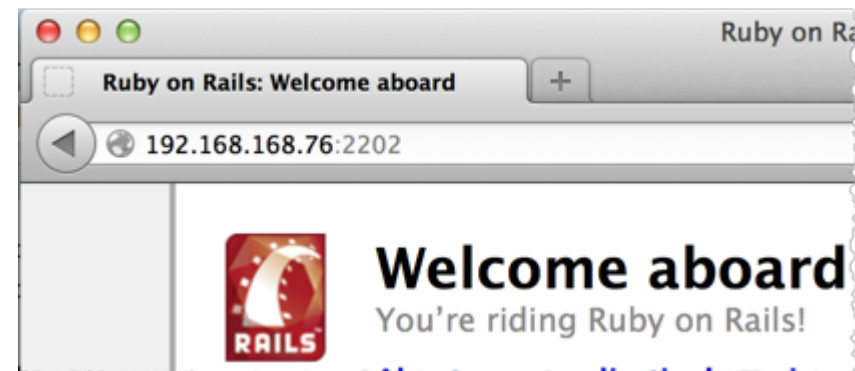
Or watch this video:
youtu.be/oc6wEDx8r5o

Application Folder Structure

```
app
  controllers
  helpers
  models
  views
    layouts
config
  environment.rb
  routes.rb
db
  database.yml
  migrations
lib
log
public
script
test
vendor
  plugins
  gems
  rails
```

Database config

`/www/A2222/htdocs/A2222/config/`**`database.yml`**

```
development:
  adapter: ibm_db
  username: A2222
  password: A2222
  database: '*LOCAL'
  schema: A2222_D

test:
  adapter: ibm_db
  username: A2222
  password: A2222
  database: '*LOCAL'
  schema: A2222_T
```
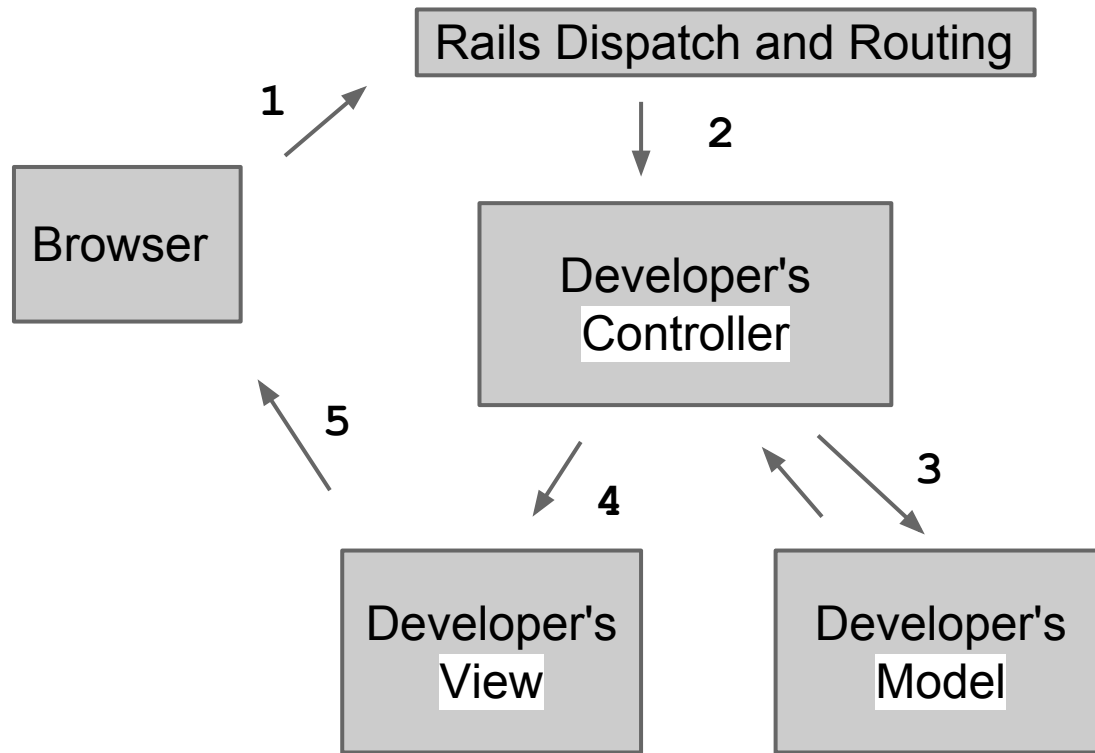
File `database.yml` stores the database connection information.

`*LOCAL` declares the Rails app is running on this machine.

There are additional options to do encryption of the clear-text password.

Request Cycle

- 10k foot view of a Rails request
- MVC is consistently encouraged

```
                          Rails Dispatch and Routing
                1                          │
                  ↗                        │ 2
                                           ↓
    ┌──────────┐              ┌─────────────────────┐
    │          │              │     Developer's     │
    │ Browser  │              │     Controller      │
    │          │              │                     │
    └──────────┘              └─────────────────────┘
          ↑                      ↙              ↖ ↘
        5 │                   4                    3
                    ┌──────────────┐      ┌──────────────┐
                    │ Developer's  │      │ Developer's  │
                    │    View      │      │    Model     │
                    └──────────────┘      └──────────────┘
```

– Rails Routing from the Outside In

Routing

The Rails router recognizes URLs and dispatches them to a controller's action. It can also generate paths and URLs, avoiding the need to hardcode strings in your views.

app/config/**routes.rb**

```
A2233::Application.routes.draw do
  resources :posts
end
```

Adding `resources :posts` to the routes.rb file creates the below paths.

HTTP Verb	Path	Action	Used for
GET	/posts	index	display list of all posts
GET	/posts/new	new	return HTML form to create new post
POST	/photos	create	create a new post
GET	/photos/:id	show	display a specific post
GET	/photos/:id/edit	edit	return an HTML form for editing a post
PATCH/PUT	/photos/:id	update	update a specific photo
DELETE	/photos/:id	destroy	delete a specific photo

guides.rubyonrails.org/routing.html – Formal Docs
mikewilliamson.wordpress.com/2012/12/12/ruby-redos-the-rails-router – Go deeper

Command:`rake routes`

Prefix - The route name. It is common to append _path and use it in a view (i.e. `edit_post_path(@post)`)

Verb - The HTTP verb associated with this path. GET is for retrieval, POST for creation of things, PUT for updating, and DELETE for destroying.

URI Pattern - Shows the pattern of the URI. The `:id` declares where the model key will be used and `:format` gives the option of specifying `.json` or `.xml`

Controller#Action - The controller and action that will receive the request.

```
$ rake routes
   Prefix Verb    URI Pattern              Controller#Action
    posts GET     /posts(.:format)         posts#index
          POST    /posts(.:format)         posts#create
 new_post GET     /posts/new(.:format)     posts#new
edit_post GET     /posts/:id/edit(.:format) posts#edit
     post GET     /posts/:id(.:format)     posts#show
          PATCH   /posts/:id(.:format)     posts#update
          PUT     /posts/:id(.:format)     posts#update
          DELETE  /posts/:id(.:format)     posts#destroy
```

ActiveRecord

RPG's extremely simple native database access has always been the one thing no other language has come close to. Then I tried Rails' ActiveRecord and fell in love.

- Map classes to tables, an Object to a Row, or ORM (Object Relation Mapping)
- Database agnostic
- Models exist in app/models
- Table names are plural and class names singular
- Database columns map to attributes (getters and setters) using Ruby's **Open Classes**
- All tables have an integer primary key, by convention, named id
- Database tables are created with migrations

```ruby
Customer.find(1)
# select * from customer where id = 1


Customer.find_by_name("Aaron Bartell")          Dynamic Finders
# select * from customer where name = 'Aaron Bartell'


Customer.find_by_date_of_birth '1979-04-22'
#select * from customer where date_of_birth = '1979-04-22'


Customer.find_by_name_and_date_of_birth 'Aaron Bartell', '1979-04-22'
# select * from customer
#       where name = 'Aaron Bartell' and date_of_birth = '1979-04-22'
```

guides.rubyonrails.org/active_record_basics.html – Formal docs
guides.rubyonrails.org/active_record_querying.html – Formal docs

ActiveRecord - DB access examples

```
Customer.count
#select count(*) as count

Customer.order 'name DESC'
#select * from customer order by name desc

Post.where 'title LIKE ?', 'p%'
#SELECT "posts".* FROM "posts" WHERE (title LIKE 'p%')
```

Create

```
user = User.new
user.first_name = "Dave"
user.last_name = "Thomas"
user.save
```

Update

```
user = User.find(12)
user.first_name = "Bill"
user.last_name = "Gates"
user.save!
```

Delete

```
User.destroy(12)
--or--
user = User.find(12)
user.destroy
```

guides.rubyonrails.org/active_record_querying.html – Formal docs

ActiveRecord - Overriding Conventions

- **Rails is very flexible!**
- Composite keys? There's a gem for that: **github.com/bosko/rmre**
- Tables already defined? "Dump" the database to a file using Rails commands.
- Table names not following plural conventions? Use `self.table_name` to alias it.
- Don't like your column names? Use `alias_attribute` to rename it.

```ruby
class MyModel < ActiveRecord::Base
  self.table_name = 'my_legacy_table'
  self.primary_key = 'my_id'
  self.pluralize_table_names = false
  alias_attribute :CSMPR, :master_proof_report
end
```

– Formal docs

ActiveRecord - Associations

Why do we need associations between models? Because they make common operations simpler and easier in your code.

Use `:through` for easily traversing the DB.

```ruby
class Patient < ActiveRecord::Base
  has_many :encounters
  has_many :physicians, :through => :encounters
end
```

Now you can do `@patient.physicians`

If you were to type out all of the various things ActiveRecord does for you then it would look like this.

Whew! Thanks to CoC!

```ruby
class Patient < ActiveRecord::Base
  has_many :encounters, :class_name => Encounter,
                        :foreign_key => 'patient_id'
  has_many :physicians, :through => :encounters,
                        :class_name => 'Physician',
                        :foreign_key => 'physician_id'
end
```

guides.rubyonrails.org/association_basics.html – Formal docs

Migrations

*Migrations are a feature of Active Record that allows you to evolve your database schema over time. **Rather than write schema modifications in pure SQL**, migrations allow you to use an **easy Ruby DSL** to describe changes to your tables.*

- Think of each migration as being a new 'version' of the database, **stored in source control for historical purposes**
- `schema.rb` contains a full representation of your database
- DB2 table `SCHEMA_MIGRATIONS` keeps track of which migrations have been run
- Migration files stored in `db/migrations`
- The `change` method is for cases where ActiveRecord knows how to reverse the migration (i.e. `add_column` reversed would be `remove_column`)

```
rails generate model Post title:string text:text
```

```ruby
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :title
      t.text :text
      t.timestamps
    end
  end
end
```

app/models/**post.rb**

```ruby
class Post < ActiveRecord::Base
end
```

guides.rubyonrails.org/migrations.html – Formal docs

Database migrations

The `rake db:migrate` command queries table `A2222_D/SCHEMA_MIGRATIONS` (`SCHEM00001` below) to learn the most recent migration run against this schema (aka library).

Recognizes `20131031191936_create_posts.rb` hasn't been run and invokes it.

```
● ● ●                    1. ssh aaron@
-bash-4.2$ rake db:migrate
==  CreatePosts: migrating =============
-- create_table(:posts)
   -> 0.3686s
==  CreatePosts: migrated (0.3688s) ====

-bash-4.2$
```
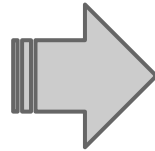
```sql
CREATE TABLE A2222_D.posts(
  id int NOT NULL
       GENERATED ALWAYS AS IDENTITY
       (START WITH 1 INCREMENT BY 1),
  title varchar(255),
  text varchar(255),
  PRIMARY KEY(id)
)
```

WRKOBJ A2222_D/*ALL

Opt	Object	Type	Library	Att
	QSQJRN0001	*JRNRCV	A2222_D	
	QSQJRN	*JRN	A2222_D	
	POSTS ←	*FILE	A2222_D	PF
	SCHEM00001	*FILE	A2222_D	PF
	SYSCHKCST	*FILE	A2222_D	LF
	SYSCOLUMNS	*FILE	A2222_D	LF

WHAT!?! Shouldn't it be **POST** and not **POST_S_**?

It is a Rails convention to name tables the plural of the model they represent.

Migrations

- Can be used to migrate data and not just structures *(see below)*
- The `up` and `down` methods are one mechanism to do custom reverts or for scenarios where "there's no going back".
- The `rake db:rollback` allows you to roll back the most recent migration if you need to correct a mistake e.g. incorrectly named a column.

```
rails generate migration UpdateProductPrice
```

```ruby
class UpdateProductPrice < ActiveRecord::Migration
  def up
    Products.connection.execute(
      'UPDATE `products` SET `price`= 1.00 WHERE 1'
    )
  end
  def down
    # No going back, dude.
  end
end
```

guides.rubyonrails.org/migrations.html – Formal docs

Controller

- The "C" in MVC, sits between model and view
- Files stored in `app/controllers`
- Gets control of processing when a request comes in
- `rails generate scaffold_controller post title:string text:text`

```ruby
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.paginate(:page => params[:page])
  end

  def show
  end

  def new
    @post = Post.new
  end

  def edit
  end

  private
    def set_post
      @post = Post.find(params[:id])
    end
```

Controller - Create

Called when the user submits a form

```ruby
class PostsController < ApplicationController
. . .
  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
        format.html {
          redirect_to @post, notice: 'Post was successfully created.' }
        format.json {
          render action: 'show', status: :created, location: @post }
      else
        format.html { render action: 'new' }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end
. . .
  private
    def post_params
      params.require(:post).permit(:title, :text)
    end
end
```

Controller - Update

Called when the user has **updated** a form for an existing model object

```ruby
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]
. . .
  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html {
          redirect_to @post, notice: 'Post was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: 'edit' }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end
. . .
  private
    def set_post
      @post = Post.find(params[:id])
    end

    def post_params
      params.require(:post).permit(:title, :text)
    end
end
```

Controller - Destroy

Called when the user select to destroy a model

```ruby
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]
. . .
  def destroy
    @post.destroy
    respond_to do |format|
      format.html { redirect_to posts_url }
      format.json { head :no_content }
    end
  end


  private
    def set_post
      @post = Post.find(params[:id])
    end
. . .
end
```

Keep your DB free from orphans with
`dependent: :destroy`
If the Post is destroyed, the comments
will also be destroyed.

```ruby
class Post < ActiveRecord::Base
  has_many :comments, dependent: :destroy
end
```

Views and Variables

```ruby
class PostsController < ApplicationController
  def index
    @posts = Post.all
  end
end
```

Request comes into controller and it prepares **instance** variables (`@posts`) that are made available to the view.

views/posts/**index.html.erb** (snippet)

The view makes use of variables and could also do additional method chaining e.g. `post. comments`

```erb
<% @posts.each do |post| %>
  <tr>
    <td><%= post.title %></td>
    <td><%= post.text %></td>
    <td><%= link_to 'Show', post %></td>
    <td><%= link_to 'Edit', edit_post_path(post) %></td>
  </tr>
<% end %>
```

guides.rubyonrails.org/layouts_and_rendering.html – Formal docs

Partials

- Hides complexity so you can see what is really going on.
- Starts with an underscore e.g. `_comment.html.erb`

views/comments/**_comment.html.erb**

```
<p>
  <strong>Commenter:</strong>
  <%= comment.commenter %>
</p>
<p>
  <strong>Comment:</strong>
  <%= comment.body %>
</p>
```

views/posts/**show.html.erb** (snippet)

```
. . .
<%= render @post.comments %>
. . .
```

Use many partials from other locations

```
<%= render "shared/ad_banner" %>
<h1>Products</h1>
<p>Here are a few of our fine products:</p>
...
<%= render "shared/footer" %>
```

guides.rubyonrails.org/layouts_and_rendering.html#using-partials – Formal docs

Form Helpers

views/posts/**_form.html.erb** (snippet)

```erb
<%= form_for(@post) do |f| %>
  <%= f.text_field :title %>
  <%= f.text_field :text %>
  <%= f.submit %>
<% end %>
```

This is what Rails generates for you. Note the `post[title]` name of the `input` field.

```html
<form accept-charset="UTF-8" action="/posts/1"
      class="edit_post" id="edit_post_1" method="post">
  <input id="post_title" name="post[title]" type="text" value="post1" />
  <input id="post_title" name="post[text]" type="text" value="post1" />
  <input name="commit" type="submit" value="Update Post" />
</form>
```

```ruby
@post = Post.new(params[:post])
@post.save
```

Now `params[:post]` will receive in all parameters that have `post[...]` surrounding them. Not only is this a big time saver but also lessens the visual complexity of the code (less lines).

guides.rubyonrails.org/form_helpers.html – aaa

Form Validation

Use `validates_precense_of` to make sure title is always occupied.

```
class Post < ActiveRecord::Base
  has_many :comments
  validates_presence_of :title
end
```

New post

1 error prohibited this post from being saved:

- Title can't be blank

Title

You can check the current instance of a model to determine if there are any errors. Errors can be manually added by your code or through things like `validates_precense_of`.

```
<%= form_for(@post) do |f| %>
  <% if @post.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@post.errors.count, "error") %>
          prohibited this post from being saved:</h2>
      <ul>
      <% @post.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
. . .
<% end %>
```

guides.rubyonrails.org/active_record_validations.html#displaying-validation-errors-in-views – Formal docs
api.rubyonrails.org/classes/ActiveModel/Errors.html – Formal docs

Many Ways to Validate

Most validations will be in the model and not in the controller. This took me awhile to get used to.

```ruby
class Patient < ActiveRecord::Base
 . . .
  validates :bio, length: { maximum: 1000,
    too_long: "%{count} characters is the maximum allowed" }

  validates_inclusion_of :gender, :in => ['male', 'female']

  validates :terms_of_service, acceptance: true
end
```

Rendering options

- Rails gives HTML and JSON rendering options out of the box
- XML can be added as a rendering option very easily
- JSON gives instant web services! BIG TIME SAVER!

The `respond_to` block allows you to respond differently depending on the request format.

```ruby
def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_url}
    format.json { head :no_content }
  end
end
```

Output from
192.168.168.76:2222/posts.**json**

```json
[
  {
    "title":"post1",
    "text":"body1",
    "url":"http://.../posts/1.json"
  },
  {
    "title":"post2",
    "text":"body2",
    "url":"http://.../posts/2.json"
  },
  {
    "title":"my title",
    "text":"my text",
    "url":"http://.../posts/3.json"
  }
]
```

guides.rubyonrails.org/layouts_and_rendering.html – Formal docs

Unit Testing

Rails makes it super easy to write your tests. It starts by generating skeleton test code and infrastructure while you are creating your models and controllers.

- Woven into the Rails fabric from the beginning
- Skeleton unit tests created when generating models and controllers.
- Simulate browser requests
- Ensure your code adheres to the desired functionality even after major code refactoring.
- Check out this video to see even more: **railscasts.com/episodes/257-request-specs-and-capybara**

Command `rake test:models` will run all model tests

```ruby
require 'test_helper'

class PostTest < ActiveSupport::TestCase
  test "should not save post without title" do
    post = Post.new
    assert !post.save
  end
end
```

guides.rubyonrails.org/testing.html – Formal docs

Fixtures in testing

- "Fixtures" is a fancy word for sample data.
- Database independent
- One file per model located in `tests/fixtures/`

Command `rake test:controllers` will run all controller tests

```
require 'test_helper'
class PostsControllerTest <
        ActionController::TestCase
  setup do
    @post = posts(:one)
  end


  test "should get edit" do
      get :edit, id: @post
      assert_response :success
  end
. . .
end
```

test/fixtures/**posts.yml**

```yaml
one:
  title: My First Post
  body: Body of my 1st post

two:
  title: My Second Post
  body: Body of my 2nd post
```

Method `setup` is run before each test and obtains the named entry from the `posts.yml` fixture and instantiates an object from it for later use.

`get` - Issues an HTTP GET request to `/posts/:id/edit`

`assert_response` - tests whether HTTP 200 OK was returned.

`assert_not_nil assigns` - makes sure a post instance variable was set.

guides.rubyonrails.org/testing.html – Formal docs

Action Mailer

...allows you to send emails from your application using mailer classes and views. Mailers work very similarly to controllers.

- Easily send HTML or plain text emails with attachments
- "Mailers" work very similar to controllers - communicate with models and render a view (html or text).

Example usage taken from `app/controllers/users_controller.rb` when the create action is called (i.e. creating a user)

```ruby
UserMailer.welcome_email(@user).deliver
```

app/mailers/**user_mailer.rb**

```ruby
class UserMailer < ActionMailer::Base
  default from: 'notifications@example.com'

  def welcome_email(user)
    @user = user
    @url  = 'http://example.com/login'
    mail(to: @user.email, subject: 'Welcome to My Awesome Site')
  end
end
```

guides.rubyonrails.org/action_mailer_basics.html – Formal docs
mailcatcher.me – Intercept email in dev and test so it never gets sent. Then you can review.

Action Mailer - Template

- Instance variables (i.e. `@user` and `@url`) are made available to the email template just like a view.

app/views/user_mailer/**welcome_email.html.erb**

```erb
<!DOCTYPE html>
<html>
  <head>
    <meta content='text/html; charset=UTF-8' http-equiv='Content-Type' />
  </head>
  <body>
    <h1>Welcome to example.com, <%= @user.name %></h1>
    <p>
      You have successfully signed up to example.com,
      your username is: <%= @user.login %>.<br/>
    </p>
    <p>
      To login to the site, just follow this link: <%= @url %>.
    </p>
    <p>Thanks for joining and have a great day!</p>
  </body>
</html>
```

guides.rubyonrails.org/action_mailer_basics.html – Formal docs

Action Mailer - Config

- Can have a different email configuration for each environment
- Make sure to use an email server that allows the volume of email you expect (GMail is limited - yes, I learned that the hard way)

config/environments/**$RAILS_ENV.rb** (i.e. development.rb)

```
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
  address:              'smtp.gmail.com',
  port:                 587,
  domain:               'example.com',
  user_name:            '<username>',
  password:             '<password>',
  authentication:       'plain',
  enable_starttls_auto: true   }
```

guides.rubyonrails.org/action_mailer_basics.html – Formal docs

Asset Pipeline

...provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages such as CoffeeScript, Sass and ERB.

- Follows "fast by default" mantra of Rails
- Saves number of `GET` requests to the server by combining all Javascript files into one
- Minifies source so you aren't sending unnecessary whitespace and comments over the internet
- Can use the Sass and CoffeeScript and have Rails compile it down to CSS and Javascript respectively
- Assets can be precompiled before deploying to production

The big long number is a fingerprint. Whenever the contents change, so does the fingerprint, thus requiring the browser to download again.

app_root/public/assets/*

```
application-853b7d1526d52c55e2557d2ba0f55806.js
application-853b7d1526d52c55e2557d2ba0f55806.js.gz
application-bb7646d9ffeacfd960a576001feb8b35.css
application-bb7646d9ffeacfd960a576001feb8b35.css.gz
```

guides.rubyonrails.org/asset_pipeline.html – Formal Docs
coffeescript.org – Learn about CoffeeScript
sass-lang.com – Learn about **S**ytactically **A**wesome **S**tyle **S**heets (SASS)

Environments

Rails environments allow a single app to behave differently based on the environment it is running in.

- Three environments by default: **development**, **test**, **production**
- Each environment can have it's own database in config/database.yml and configuration file under config/environments
- `ENV["RAILS_ENV"]` defines the environment
- Add custom environments (i.e. staging - just like production except it maybe uses a test credit card gateway account vs. a real one)

Good example of needing a **custom environment**… if your site takes credit cards it is good to have an **environment exactly like production** but doesn't use the production credit card gateway account. Create a new environment named **"staging"** to accomplish this.

app/config/environments/**development.rb**

```
config.cache_classes = false
```

Code is reloaded on every request in the development environment. This slows down response time but is perfect for development since you don't have to restart the web server when you make code changes.

i18n (internationalization)

The Ruby i18n framework provides you with all necessary means for internationalization/localization of your Rails application.

- Problem is complex, so Rails does the following:
  - providing support for English and similar languages out of the box
  - making it easy to customize and extend everything for other languages
- All static strings in Rails framework - e.g. ActiveRecord validation messages - have been internationalized.
- There are **many** options Rails offers to solve this challenge - since there are many different needs (i.e. some desire domain.es and some domain.com?locale=es)

Sample URL: http://domain.com?**locale=pt**

One approach, use yaml files
config/locales/**en.yml**

controllers/**application_controller.rb**

```ruby
before_action :set_locale

def set_locale
  I18n.locale = params[:locale] ||
  I18n.default_locale
end
def default_url_options(options={})
  { locale: I18n.locale }
end
```

```yaml
en:
  hello: "Hello world"
```

guides.rubyonrails.org/i18n.html – Formal Docs

Rails Console

… lets you interact with your Rails application from the command line.

- Loads your application into an environment similar to starting the web server
- Use `rails c` for short
- Uses `irb` under the covers
- Great for quickly testing, debugging, or just trying things out
- Query the database with ActiveRecord statements (i.e. `Post.find(1)`)

Example: List all methods of a model object containing "has"

```
rails> Post.methods.grep /^has/
 => [:has_secure_password, :has_many, :has_one, :has_and_belongs_to_many, :hash]
```

Example: Learn what a model's route looks like

```
rails> app.post_path Post.first
  Post Load (0.2ms)  SELECT "posts".* FROM ...
 => "/posts/1"
```

Example: Learn what a model's `link_to` looks like

```
rails> helper.link_to 'Show', app.post_path(post)
 => "<a href="/posts/1">Show</a>"
```

guides.rubyonrails.org/command_line.html#rails-console – aaa

we Have ReacHeD THe enD!

Aaron Bartell

abartell@krengeltech.com - @aaronbartell

LITMIS.COM - Open Source for i