Ruby on Rails 5 Essential Training

Test Server: test ruby campground

XXX.XXX.XXX.XXX

<Username> / <Password>

info

rails app root directory (AppRoot): /home/<Username>/Sites/simple\_cms

mysql username: rails\_user

password: <Password>

databases: simple\_cms\_development

Loaded Gems: AppRoot/gemfile

Gems configuration directory: AppRoot/config/\*.yml

Log directory: AppRoot/log/

Requirements

Ruby 5

MySQL

Main commands

//\* start the app on port 3000

$ cd /home/<Username>/Sites/simple\_cms

//\* go to AppRoot

$ rails server

MYSQL

username: root password: <Password>

username: rails\_user password: <Password>

COPYING EXERCISE FILES

//\* copy all chapters directories to c:\temp\simple\_cms\_exercises

C:> pscp -r -P 2202 c:\tmp\simple\_cms [<Username>@ XXX.XXX.XXX.XXX:\tmp](mailto:%3cUsername%3e@XXX.XXX.XXX.XXX:\tmp)

$ cd /home/<Username>

$ mkdir Sites

$ cd Sites

$ mv /tmp/simple\_cms \_exercises /home/<Username>/

CREATE DATABASE AND MYSQL USER

$ mysql -u root -p

//\* <Password>

mysql> create database simple\_cms\_development;

mysql> create database simple\_cms\_test;

mysql> create user 'rails\_user'@'localhost' identified by '<Password>';

mysql> grant all privileges on simple\_cms\_development.\* to 'rails\_user'@'localhost' identified by '<Password>';

mysql> grant all privileges on simple\_cms\_test.\* to 'rails\_user'@'localhost' identified by '<Password>';

CONFIGURING RAILS TO USE MYSQL

$ nano AppRoot/config.database.yml

username: rails\_user

password: <Password>

$ rails db:schema:dump

//\* test by dumping mysql shcema to a text file (AppRoot/db/ schema.rb). If no errors, then

database connection works

IMPORTING EXERCISES FILES

$ cd /home/<Username>/Sites/simple\_cms

$ rails otl:import

Using exercises files:

1. Copy the folder from the exercise directory to the “Sites” directory in the app root folder.

I.E. copying “Chapter 06/06\_10/simple\_cms” to “App root”/Site/simple\_cms

1. $ cd “App root”/ Sites/simple\_cms/
2. Rake otl:import

It runs an import script

Ruby: It is an object-oriented language

Rails: a web application framework written in Ruby

Framework: a set of code libraries and data structures that provide functionality

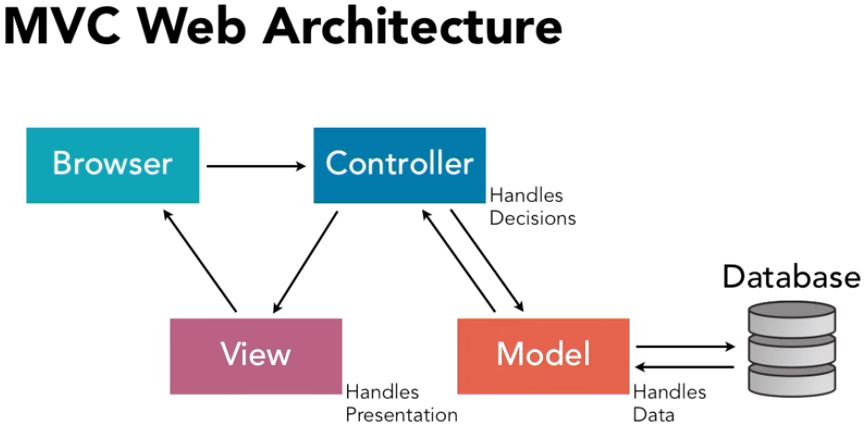
Ruby Principles:

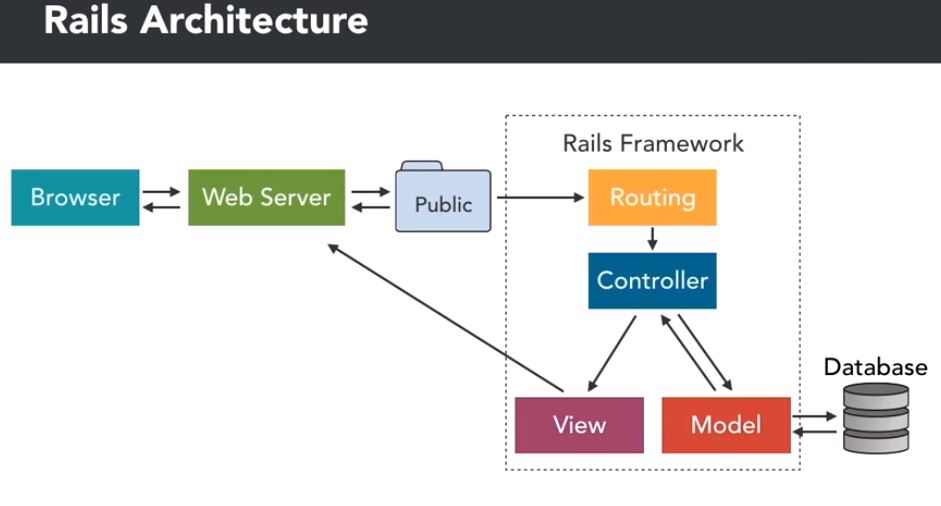
1. DRY Code: Don’t Repeat Yourself.
2. Concise, consistent code that is easy to maintain

MVC Architecture used by Ruby

Model, View, and Controller

1. Model: Data objects design
2. View: Presentation layer (HTML, CSS, etc.)
3. Controller: Processes response to user events (links, click, etc.)





Rail Architecture Explained:

1. A browser requests a file(page) to the webserver (puma, apache, nginx).
2. The webserver looks for the page in the public directory and transmits it back to the browser
3. If the file is not found in the public directory, the webserver asks the rails framework to handle the request.

3.1) The request is sent to the “Routing” to determine which “controller” to use

3.2) The controller contacts the model to get data and orders the view to transmit the data back

to the browser.

Passenger: it is a ruby module that allow rails to talk to apache

Bundler: it is a ruby gem that load all gems used by the application. It is normally used to update a when changes are made on the gems. Bundler reads and writes the following 2 files.

AppRoot/Gemfile file

//\* it contains what gems will be loaded for the application

//\* edit if you want to change the version of a gem that loads by bundler

//\* it can be edited to disable some gems from the application (they don’t load)

//\* after editon Gemfile, run bundler install

AppRoot/Gemfile.lock file

//\* it is the manifest (log) of bundler. It storage all gems version loaded

//\* never edit this file

$ bundle install

//\* it rebuilt the AppRoot/Gemfile.lock by reading & processing the AppRoot/Gemfile , updating

all gems with correct versions

//\* It must be run when a changes is made on AppRoot/gemfile

//\* it is useful if the application does not start or if mysql, etc was upgraded

$ bundler exec rake:migrate

//\* run a command in the application

//\* it runs the command in the context/environment of the application.

Create an Application:

$ cd /home/<Username>

$ rails new simple\_cms -d mysql

//\* creates a new app that uses MySQL

The AppRoot is /home/<Username>/simple\_cms/

Accessing the Application/Project

START THE WEB SERVER

$ cd /home/<Username>/Sites/simple\_cms

//\* go to the AppRoot

$ rails server

$ rails s

//\* start the web server

//\* localhost:3000

Error: Mysql2::Error::ConnectionError (Access denied for user 'root'@'localhost' (using password: NO)):

$ nano /home/<Username>/simple\_cms/config/database.yml

Password: <Password>

Error: simple\_cms database does not exist

Development

Database: #simple\_cms

$ rails server

http://localhost:3000

RAILS GENERATE

The rails generate command uses templates to create a whole lot of things. Running rails generate by itself gives a list of available generators.

You can install more generators through generator gems, portions of plugins.

$ rail generate

//\* show a list of available generators that can be installed

GENERATE A CONTROLLER AND A VIEW

Rails generate controller: stubs out a new controller and its views as arguments to this command.

$ rails generate controller demo index

//\* creates a controller named demo, and one view named index

$ nano AppRoot/app/controllers/demo\_controller.rb

//\* show actions. In this example, the action is to show the dem view

layout false

//\* turns layout off

def index

$ nano AppRoot/app/views/demo/index.html.erb

<h1>Demo#index</h1>

<p> Hello world! </p>

$ rails server

http://localhost:3000/demo/index

//\* XXX.XXX.XXX.XXX:3000/demo/index

LIST ALL VIEWS:

Nano AppRoot/config/routes.rb

ROUTE

It allows a specific controller to handle http request by specifying available controllers and their location

Notation:

:Controller/:View/:id

Routes Types:

1. Simple route

A specific route defined in AppRoot/config/routes.rb

get ‘demo/index’

1. Default route

A generic route defined in AppRoot/config/routes.rb

It does not specify each route, but tell the controller to server any controller/view/id combo if it exist

get ‘:controller(/:action(/:id))’

1. Root route

A route that handles the HtmlRoot. It would change what is shown in

http://localhost:3000

It can be defined in AppRoot/config/routes.rb

Root “demo#index”

1. Resourceful routes

MANUALLY CREATE A VIEW

$ nano AppRoot/app/views/demo/hello.html.erb

<h1>Hello wordl from Hello.html.erb<h1>

$ nano AppRoot/config/routes.rb

get ‘demo/hello’

//\* creates a simple route for hello

$ nano AppRoot/app/controllers/demo\_controller.rb

def hello

end

//\* specifies an action. If empty, it will follow the default action, which is

render(‘hello’)

$ cd AppRoot

$ rails server

http://localhost:3000/demo/hello

TEMPLATES:

A view that will have dynamic content added.

It is done by modifying the default action in AppRoot/app/controllers/demo\_controller.rb

$ nano AppRoot//app/controllers/demo\_controller.rb

def hello

render(‘index’)

//\* it will forward any request from the hello view to the index view.

end

REDIRECTS:

When a controller redirects the action to another controller, internal or external

Rails do it using HTTP redirection. So the browser url changes

Creating a redirect

$ nano AppRoot/config/routes.rb

get ‘demo/other\_hello’

//\* creates a route

get ‘demo/google’

//\* creates a route

$ nano AppRoot/app/controllers/demo\_controller.rb

def other\_hello

redirect\_to(:controller => 'demo', :action => 'index')

//\* it will forward any request from the other\_hello view to the index view.

end

def google

redirect\_to(‘http://www.google.com’)

end

$ cd AppRoot

$ rails server

http://localhost:3000/demo/other\_hello

<http://localhost:3000/demo/google>

3. Controllers, Views, and Dynamic Content

Redirect actions

ERB

Aka Embedded Ruby.

It allows ruby code to run and output html, jscript, json, etc

ERB File

A file that contains ruby code

Example: AppRoot/app/controllers/demo\_controller.rb

Inserting Ruby code inside an ERB file using ERB tags:

<%code %>

//\* it processes and executes the ruby code

<%= code %>

//\* it processes the ruby code and present the output the results into the template

//\* similar to php

$ nano AppRoot/app/views/demo/hello.html.erb

<% 1 + 1 %>

//\* calculates 1+1

<%= 2 + 2 %>

//\* calculates 2+2 and output the results to the view

<% target = “world” %>

//\* Declares a variable and give it a value of “world”

<br>

<%= “Hello #{target}” %>

//\* Outputs “Hello world”

//\* #{x} replace x by its value

$ cd AppRoot

$ rails server

http://localhost:3000/demo/hello

//\* only 4 is shown

Another example:

$ nano AppRoot/app/views/demo/hello.html.erb

//\* this will show only the last value of n

<br />

<%=

3.times do |n|

//\* loop similar to for n=0 to 2;n++

puts n

//\* it output the result of n into the console, not the template

%>

//\* this will each value of n in the loop

<% 5.times do |n| %>

<%= n %>

<% end %>

$ cd AppRoot

$ rails server

http://localhost:3000/demo/hello

INSTANCE VARIABLES

Regular variable

x

Instance variable: it is a variable local to the current instance (scope, environment). The instance is

defined by the controller.

@x

$ nano AppRoot/app/controllers/demo\_controller.rb

Class DemoController, defines the class or instance

def hello

@array = [1,2,3,4,5]

Render(‘hello’)

End

$ nano AppRoot/app/views/demo/hello.html.erb

<% @array.each do |n| %>

<%= n %>

<br/>

<% end %>

LINKS

<%= link\_to(“text”, target) %>

//\* the output of this is: <A HREF=target> text </A>

Target can be

* url
* ruby hash

{:controller => ‘demo’, :action => ‘index’}

Rails uses the route.erb file to convert the ruby hash into the target url

$ nano AppRoot/app/views/demo/index.html.erb

<%= link\_to("link to ", {:action => 'hello'} ) %>

URL PARAMETERS

Parameters are strings

$ nano AppRoot/app/views/demo/index.html.erb

<%= link\_to("link to with parameters", {:action => 'hello', :page => 5, :id => 20} ) %>

//\* this will be translated to

<A HREF=’localhost:3000/demo/hello?id=20&page=5> text </A>

$ nano AppRoot/app/views/demo/hello.html.erb

<% @id = params['id'] %>

<br /> Id:

<%= @id %>

<br/> page:

<%= params[:page] %>

<br/> Next page:

<% @page = params[:page] %>

<%= @page.to\_i + 1 %>

//\* to\_i converts a string to numeric

$ cd AppRoot

$ rails server

<http://localhost:3000/demo/hello>

However, it is a bad idea to use the params[] inside a view, because an error will occur if the view is called without parameters. It is better to add params[] in the controller

$ nano AppRoot/app/controllers/demo\_controller.rb

def hello

@id = params[‘id’]

@page = params[:page]

//\* ‘page’ or :page can be used, the first is a string, the second is a symbol

End

To control the other of parameters in the final url, you will need to specify the order in routes.erb for the controller and view

$ nano AppRoot/config/routes.erb

get ‘demo/hello/:id’

DATABASES AND MIGRATIONS

CRUD: Create, Read, Update, and Delete.It is the way ruby interact with databases

Configure rails to connect to a databases:

$ nano AppRoot/config/database.yml

Update username and password

Migration: It is a set of database instructions, like a script but at schema level

It can be used to

* Move up to a new state. Implement changes
* Move back to the previous state. Restore changes
* Create versions of a database
* Use Ruby to interact with database instead of using SQL

Create a Migration

//\* creates a migration template (empty), that can be modified to add code

$ cd AppRoot

$ rails generate migration MigrationSample

//\* it creates a file in AppRoot/db/migrtation/<timestampt>MigrationSample.rb

$ nano AppRoot/db/migrtation/<timestampt>MigrationSample.rb

Classs MigrationSample < ActiveRecord::Migration[5.0]

def up

//\* code that will be executed when migrating up

end

def down

//\* code that will be executed when migrating down

//\* as a standard. This code should revert the changes defined in

//\* the “up” method. In revert order as in the “up” method

end

Generate Models:

Models are similar to Migration but used to define tables. it contains the model definition for tables. In other words, it defines tables within the database as opposed to Migration which defines databases only.

Rails automatic creates a column named “id” that identifies each record (primary key)

$ cd AppRoot

$ rails generate model User

//\* it creates a migration file in AppRoot/db/migrate/<timestamp>create\_users.rb

//\* It creates a model definition file in AppRoot/app/models/user.rb

$ nano AppRoot/db/migrate/<timestamp>create\_users.rb

class CreateModelSamples < ActiveRecord::Migration[5.2]

def up

create\_table :model\_samples do |t|

t.column "first\_name", :string, :limit => 25

t.column "last\_name", :string, :limit => 50

t.column "email", :string, :default => '', :null => false

t.column "password", :string, :limit => 40

t.datetime “created\_at”

t.datetime “updated\_at”

//\* if column name if “created\_at” or “updated\_at”, rails will

//\* automatic populate their values when a new record is

//\* created

t.timestamps

//\* this is equivalent to the previous 2 definitions “created\_at”

//\* and “updated\_at”

end

def down

drop\_table :users

end

$ mysql -u rails\_user -p

//\* <Password>

mysql> use simple\_cms\_development;

mysql> show tables;

mysql> select \* from model\_samples;

mysql> show fields from model\_samples;

Types of Table’s Column

binary

boolean

date

datetime

if column name if “created\_at” or “updated\_at”, rails will automatic populate their

values when a new record is created

decimal

float

integer

string

text

time

Options for Table’s Column

:limit => size

//\* how large a string or a number can be,

:default => value

//\* set a default value for the column

:null => true/false

//\* allow or disallow null values for the column

:precision => number

//\* for decimal type, it set the precision

:scale => number

//\* for decimal type, it set the scale

Run a Migration

$ rails db:migrate

//\* run all the migrations.

//\* useful after a new migration was created to keep database updated

Show current migration version

$ nano ./db/schema.rb

Or

mysql> select \* from schema\_migrations;

Remove Migrations

$ rails db:migrate VERSION=0

//\* undo all previous run migrations, and goes back to version 0 (first one)

$ rails db:migrate VERSION=201806276230

//\* performs a partial restore

//\* undo all changes and leaves the databases as in the migrations with ID:

//\* 201806276230

//\* you can get the migration id from ./db/schema.rb or from the table

//\* schema\_migrations;

Run a UP or Down method from a migration without removing

$ rails db:migrate:up VERSION=201806276230

//\* it runs the up method specified in the migration 201806276230

//\* it does not remove any migration

$ rails db:migrate:down VERSION=201806276230

$ rails db:migrate:redo VERSION=201806276230

It does a down, then it does an up method

Migration Methods:

This are methods that can be used inside the migration file (AppRoot/db/migrate/<timestamp>name)

Table Migration Methods:

create\_table(table, options) do |t|

…columns…

drop\_table(table)

rename\_table(table, new name)

Column Migration Methods

add\_column(table, column, type, options)

remove\_column(table, column)

rename\_column(tabe, column, new\_name)

change\_column(table, column, type, options)

Index Migration Methods

Add\_index(table, column, options)

Options:

:unique => true/false

:name => “your\_custom\_name”

Remove\_index(table ,column)