== Welcome to ETH-N Collection

Rails is a web-application framework that includes everything needed to create

database-backed web applications according to the Model-View-Control pattern.

This pattern splits the view (also called the presentation) into "dumb" templates

that are primarily responsible for inserting pre-built data in between HTML tags.

The model contains the "smart" domain objects (such as Account, Product, Person,

Post) that holds all the business logic and knows how to persist themselves to

a database. The controller handles the incoming requests (such as Save New Account,

Update Product, Show Post) by manipulating the model and directing data to the view.

In Rails, the model is handled by what's called an object-relational mapping

layer entitled Active Record. This layer allows you to present the data from

database rows as objects and embellish these data objects with business logic

methods. You can read more about Active Record in

link:files/vendor/rails/activerecord/README.html.

The controller and view are handled by the Action Pack, which handles both

layers by its two parts: Action View and Action Controller. These two layers

are bundled in a single package due to their heavy interdependence. This is

unlike the relationship between the Active Record and Action Pack that is much

more separate. Each of these packages can be used independently outside of

Rails. You can read more about Action Pack in

link:files/vendor/rails/actionpack/README.html.

== Getting Started

1. At the command prompt, start a new Rails application using the <tt>rails</tt> command

and your application name. Ex: rails myapp

2. Change directory into myapp and start the web server: <tt>script/server</tt> (run with --help for options)

3. Go to http://localhost:3000/ and get "Welcome aboard: You're riding the Rails!"

4. Follow the guidelines to start developing your application

== Web Servers

By default, Rails will try to use Mongrel and lighttpd if they are installed, otherwise

Rails will use WEBrick, the webserver that ships with Ruby. When you run script/server,

Rails will check if Mongrel exists, then lighttpd and finally fall back to WEBrick. This ensures

that you can always get up and running quickly.

Mongrel is a Ruby-based webserver with a C component (which requires compilation) that is

suitable for development and deployment of Rails applications. If you have Ruby Gems installed,

getting up and running with mongrel is as easy as: <tt>gem install mongrel</tt>.

More info at: http://mongrel.rubyforge.org

If Mongrel is not installed, Rails will look for lighttpd. It's considerably faster than

Mongrel and WEBrick and also suited for production use, but requires additional

installation and currently only works well on OS X/Unix (Windows users are encouraged

to start with Mongrel). We recommend version 1.4.11 and higher. You can download it from

http://www.lighttpd.net.

And finally, if neither Mongrel or lighttpd are installed, Rails will use the built-in Ruby

web server, WEBrick. WEBrick is a small Ruby web server suitable for development, but not

for production.

But of course its also possible to run Rails on any platform that supports FCGI.

Apache, LiteSpeed, IIS are just a few. For more information on FCGI,

please visit: http://wiki.rubyonrails.com/rails/pages/FastCGI

== Apache .htaccess example

# General Apache options

AddHandler fastcgi-script .fcgi

AddHandler cgi-script .cgi

Options +FollowSymLinks +ExecCGI

# If you don't want Rails to look in certain directories,

# use the following rewrite rules so that Apache won't rewrite certain requests

#

# Example:

# RewriteCond %{REQUEST\_URI} ^/notrails.\*

# RewriteRule .\* - [L]

# Redirect all requests not available on the filesystem to Rails

# By default the cgi dispatcher is used which is very slow

#

# For better performance replace the dispatcher with the fastcgi one

#

# Example:

# RewriteRule ^(.\*)$ dispatch.fcgi [QSA,L]

RewriteEngine On

# If your Rails application is accessed via an Alias directive,

# then you MUST also set the RewriteBase in this htaccess file.

#

# Example:

# Alias /myrailsapp /path/to/myrailsapp/public

# RewriteBase /myrailsapp

RewriteRule ^$ index.html [QSA]

RewriteRule ^([^.]+)$ $1.html [QSA]

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^(.\*)$ dispatch.cgi [QSA,L]

# In case Rails experiences terminal errors

# Instead of displaying this message you can supply a file here which will be rendered instead

#

# Example:

# ErrorDocument 500 /500.html

ErrorDocument 500 "<h2>Application error</h2>Rails application failed to start properly"

== Debugging Rails

Sometimes your application goes wrong. Fortunately there are a lot of tools that

will help you debug it and get it back on the rails.

First area to check is the application log files. Have "tail -f" commands running

on the server.log and development.log. Rails will automatically display debugging

and runtime information to these files. Debugging info will also be shown in the

browser on requests from 127.0.0.1.

You can also log your own messages directly into the log file from your code using

the Ruby logger class from inside your controllers. Example:

class WeblogController < ActionController::Base

def destroy

@weblog = Weblog.find(params[:id])

@weblog.destroy

logger.info("#{Time.now} Destroyed Weblog ID ##{@weblog.id}!")

end

end

The result will be a message in your log file along the lines of:

Mon Oct 08 14:22:29 +1000 2007 Destroyed Weblog ID #1

More information on how to use the logger is at http://www.ruby-doc.org/core/

Also, Ruby documentation can be found at http://www.ruby-lang.org/ including:

\* The Learning Ruby (Pickaxe) Book: http://www.ruby-doc.org/docs/ProgrammingRuby/

\* Learn to Program: http://pine.fm/LearnToProgram/ (a beginners guide)

These two online (and free) books will bring you up to speed on the Ruby language

and also on programming in general.

== Debugger

Debugger support is available through the debugger command when you start your Mongrel or

Webrick server with --debugger. This means that you can break out of execution at any point

in the code, investigate and change the model, AND then resume execution!

You need to install ruby-debug to run the server in debugging mode. With gems, use 'gem install ruby-debug'

Example:

class WeblogController < ActionController::Base

def index

@posts = Post.find(:all)

debugger

end

end

So the controller will accept the action, run the first line, then present you

with a IRB prompt in the server window. Here you can do things like:

>> @posts.inspect

=> "[#<Post:0x14a6be8 @attributes={\"title\"=>nil, \"body\"=>nil, \"id\"=>\"1\"}>,

#<Post:0x14a6620 @attributes={\"title\"=>\"Rails you know!\", \"body\"=>\"Only ten..\", \"id\"=>\"2\"}>]"

>> @posts.first.title = "hello from a debugger"

=> "hello from a debugger"

...and even better is that you can examine how your runtime objects actually work:

>> f = @posts.first

=> #<Post:0x13630c4 @attributes={"title"=>nil, "body"=>nil, "id"=>"1"}>

>> f.

Display all 152 possibilities? (y or n)

Finally, when you're ready to resume execution, you enter "cont"

== Console

You can interact with the domain model by starting the console through <tt>script/console</tt>.

Here you'll have all parts of the application configured, just like it is when the

application is running. You can inspect domain models, change values, and save to the

database. Starting the script without arguments will launch it in the development environment.

Passing an argument will specify a different environment, like <tt>script/console production</tt>.

To reload your controllers and models after launching the console run <tt>reload!</tt>

== dbconsole

You can go to the command line of your database directly through <tt>script/dbconsole</tt>.

You would be connected to the database with the credentials defined in database.yml.

Starting the script without arguments will connect you to the development database. Passing an

argument will connect you to a different database, like <tt>script/dbconsole production</tt>.

Currently works for mysql, postgresql and sqlite.

== Description of Contents

app

Holds all the code that's specific to this particular application.

app/controllers

Holds controllers that should be named like weblogs\_controller.rb for

automated URL mapping. All controllers should descend from ApplicationController

which itself descends from ActionController::Base.

app/models

Holds models that should be named like post.rb.

Most models will descend from ActiveRecord::Base.

app/views

Holds the template files for the view that should be named like

weblogs/index.html.erb for the WeblogsController#index action. All views use eRuby

syntax.

app/views/layouts

Holds the template files for layouts to be used with views. This models the common

header/footer method of wrapping views. In your views, define a layout using the

<tt>layout :default</tt> and create a file named default.html.erb. Inside default.html.erb,

call <% yield %> to render the view using this layout.

app/helpers

Holds view helpers that should be named like weblogs\_helper.rb. These are generated

for you automatically when using script/generate for controllers. Helpers can be used to

wrap functionality for your views into methods.

config

Configuration files for the Rails environment, the routing map, the database, and other dependencies.

db

Contains the database schema in schema.rb. db/migrate contains all

the sequence of Migrations for your schema.

doc

This directory is where your application documentation will be stored when generated

using <tt>rake doc:app</tt>

lib

Application specific libraries. Basically, any kind of custom code that doesn't

belong under controllers, models, or helpers. This directory is in the load path.

public

The directory available for the web server. Contains subdirectories for images, stylesheets,

and javascripts. Also contains the dispatchers and the default HTML files. This should be

set as the DOCUMENT\_ROOT of your web server.

script

Helper scripts for automation and generation.

test

Unit and functional tests along with fixtures. When using the script/generate scripts, template

test files will be generated for you and placed in this directory.

vendor

External libraries that the application depends on. Also includes the plugins subdirectory.

If the app has frozen rails, those gems also go here, under vendor/rails/.

This directory is in the load path.