Ruby on Rails

Ruby on Rails is a Web application framework for Ruby. It was first released to the public in July 2004.

Within months, it was a widely used development environment. Many multinational corporations are using it to create Web applications.

It is the standard Web-development framework for Ruby.

Model/View/Controller

All Rails applications are implemented using the Model/View/Controller (MVC) architecture.

Models are objects that represent the components of an application that perform information processing in the problem domain.

Models should represent “real world” entities:
- physical entities like a valve in a control system, or
- conceptual entities like a department in an office, or a contract between two businesses.

Views are objects that display some aspect of the model. They are the output mechanism for the models.

You could have a view that represents:
- the position of the valve or the temperature of a chemical vat (graphical)
- cells in a spreadsheet (tabular)
- the terms and conditions of a contract (textual)
- product installation instructions (video or audio)

Controllers are objects that control how user actions are interpreted. They are the input mechanism for the views and models.

For example, the interpretation of a double-click on a temperature gauge would be handled by the controller notifying the model in a way it agrees to respond to.
MVCs come in a triad, with communication between the components occurring as follows:

The primary communication path is from Model to View. Whenever the state of a model changed, the view would need to display itself differently.

For instance, consider a View that represents a car's engine temperature.

If the engine temperature goes up (say, because a fan belt breaks) the gauge showing the temperature will need to redisplay its new value in response to that change.

Exercise: Come up with an example of MVC, specifying what’s in the model, what’s in the view, and what’s in the controller. You may work with your neighbors in doing this. Then submit your example via the form linked to the announcements on the course Web site.

In Rails, when a project is created, it is given folders for model, view, and controller classes.

A Rails application accepts an incoming request from a Web page, then gives it to a router. The router parses the URL and eventually directs the request to a particular controller.

For example, the URL might be something like http://expertiza.ncsu.edu/users/show/554.

This means to show the various fields in the entry for User 554 (name, e-mail address, role). In this case,
• the controller is **users**,  
• the action is **show**, and  
• the ID of the user is **108**.

Or, we might have something like  

What do you think this would represent?  
**Adding product 353 to the user's cart.**

What do you think are the advantages of using an MVC architecture?

---

**Downloading Rails to your computer**

Assuming that you have already installed Eclipse and RDT, you can proceed directly to the installation of Rails.

Visit [http://rubyforge.org/projects/instantrails](http://rubyforge.org/projects/instantrails) and download **InstantRails-2.0-win.zip**.

Then extract it to a directory whose name **has no spaces**. I use C:\InstantRails.

Then find the InstantRails icon in this directory, and double-click on it. The InstantRails window appears.

The first time you do this, InstantRails will ask if you want to update the paths in all of the configuration files. Say yes.

To start a Rails server, you need to click on the little black “I” icon near the upper-left corner. Select “Rails Applications > Open Ruby Console Window.”
Then a command window will pop up. The working directory will be C:\InstantRails\rails_apps (or whatever path you specified for the installation, followed by “\rails_apps”)

This Ruby console can be used to generate scripts to create database tables, and to start the server (once an application has been created).


**The Cookbook Application**

To bring up the Cookbook application, create a new Ruby project (right-click in Package Explorer view, then New Project.

We name the project “cookbook” and then deselect the “Use default location” checkbox.

This allows us to use the cookbook code that was downloaded with InstantRails, instead of a new copy of that code. Why is this important?
We find the downloaded code in C:\InstantRails\rails_apps\cookbook.

Click “OK” and then “Finish”.

Then launch InstantRails.

Click the little beige “I” box in the upper right-hand corner, and choose “Rails Applications > Open Ruby Console Window”. This gives you a terminal window:
Type “cd cookbook”, then “ruby script/server”. This starts the Web server.

C:\InstantRails\rails_apps>cd cookbook
C:\InstantRails\InstantRails\rails_apps\cookbook>ruby script/server

It will respond with this:

=> Booting WEBrick …
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shut down server; call with --help for options
** Starting Mongrel listening at 0.0.0.0:3000
** Starting Rails with development environment…
Then we can bring up a Web browser and browse to http://localhost:3000, which redirects to http://localhost:3000/recipe/list.

Now, by browsing to http://localhost:3000, we can exercise the Cookbook application.

The Cookbook application
Let’s tour the built-in Cookbook application.
Browse the structure of the Cookbook project. The app directory is where the application is stored. It has four subdirectories:

- The controllers subdirectory contains the controller classes. Each controller is responsible for handling a certain type of input from the user.
- The views subdirectory holds the display templates that the application will use to interact with the user. These files are mostly HTML, with small amounts of Ruby code embedded (ERb code). The Ruby code is executed to create a browser screen for the user.
- The models subdirectory holds the “business logic” of our application. It uses the data in our application's database.
- The helpers subdirectory holds any “helper” classes that are needed by the model and controller classes, but especially the view classes. They contain functionality needed by several view classes, and help keep program logic out of the views.

But before we consider the code here, let’s walk through the database.

Our database has two tables: categories and recipes.

- The categories table just has an ID and a name.
- The recipes table has a few additional fields, title, description, date, instructions, and category_id.

Now let’s go through the files in the application.

my_test_controller
Let’s start here, because this provides the most basic functionality. It is a “Hello, world” program. The “index” function is invoked when the my_test
controller is referenced without an action being specified (index is the default).

application_controller
This controller is essentially empty in the cookbook application. Only the auto-generated stub exists.

Methods defined in this controller are available to all controllers, because other controllers are subclasses of this controller.

For example, one might put an authentication method here and invoke it from all other controllers (e.g., as a before_filter :authorize).

category_controller
This is a perfect example of the functionality of Rails scaffolds, because it uses the pre-existing scaffold unchanged.

Scaffolding is a quick way to bring up bare-bones functionality for a database table. It is explained at http://guides.rubyonrails.org/getting_started.html#getting-up-and-running-quickly-with-scaffolding [Try to destroy a category that has instances of it]

Note the notation “:category”. This denotes a symbol.

A symbol looks like a variable name preceded by a colon. Two ways of looking at them …

- string literals that act like constants
- “the thing named” category.

recipe_controller
This doesn’t use the scaffold function, but rather creates its own functions.
Models
Relationships in models

A relationship may be

- one-to-one (e.g., a course has a syllabus and a syllabus belongs to one course).
  
  o class Syllabus < ActiveRecord::Base
    belongs_to :course
    # …
  end

  o class Course < ActiveRecord::Base
    has_one :syllabus
    # …
  end

- one-to-many (e.g., a course has many assignments)
  
  o class Assignment < ActiveRecord::Base
    belongs_to :course
    # …
  end

  o class Course < ActiveRecord::Base
    has_many :assignments
    # …
  end

- many-to-many (e.g., a course has many students; students have many courses)
  
  o class Student < ActiveRecord::Base
    has_and_belongs_to_many :courses
# …
end

  o class Course < ActiveRecord::Base
     has_and_belongs_to_many :students
     # …
   end

**category.rb**
Based on the above discussion, what should be in this class?

**recipe.rb**
Based on the above discussion, what should be in this class?

**standard_layout.rhtml**
Provides a header and various links that appear on the bottom of each page. It is referenced at the beginning of each controller that uses it.

    <%= link_to "Create new recipe",
    :controller => "recipe",
    :action => "new" %>

First parameter is the text to be displayed in the hyperlink.

(If no controller is specified, the current one is used.)