

Pengenalan Pemrograman Ruby

Fajar Muharandy

muharandy@yahoo.com

Lisensi Dokumen:

Copyright © 2003-2007 IlmuKomputer.Com

Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Tutorial ini merupakan pengenalan awal bagi anda yang belum pernah melakukan pemrograman dengan Ruby. Namun, hal-hal yang diajarkan di dalam tutorial ini tidak akan membahas konsep-konsep dalam bahasa Ruby secara detail. Malah, bisa saya bilang bahwa tutorial ini lebih tepat dijadikan “pendamping” bagi anda yang ingin membaca tutorial *Pengenalan Ruby on Rails*¹, ketimbang menjadikan tutorial ini sebagai referensi untuk belajar Ruby secara menyeluruh.

Apa yang saya tulis di sini kebanyakan merupakan hal-hal dimana saya *tersandung* ketika mempelajari Ruby di awal. Sehingga, saya harap pengalaman saya ini dapat dijadikan pelajaran berharga bagi anda semua.

Penulis

¹ Saya juga menulis tutorial *Pengenalan Ruby on Rails*, yang bisa anda dapatkan di website tempat anda mendownload tutorial ini juga.

Daftar Isi

Daftar Isi	2
1. Variable	3
2. Array dan Hash	4
3. Control Structure	6
4. Method	7
5. Classes	8
6. Kemana Setelah Ini	10
Lampiran A: Menginstal Ruby Pada Windows	11
Lampiran B: Biografi Penulis	13

1. Variable

Berbeda dengan bahasa pemrograman seperti C dan Java, di dalam Ruby kita dapat langsung mendefinisikan sebuah variable tanpa menentukan tipenya. Anda dapat mencobanya langsung melalui *irb*².

```
$irb> a = 2
$irb> b = 2
$irb> a + b
$=> 4
```

Bukan hanya tipe data numeric, anda juga bisa membuat tipe data string secara langsung.

```
$irb> a = "hehe"
$irb> a.length
$=> 4

$irb> a.reverse
$=> "eheh"

$irb> a
$=> "hehe"
```

Di dalam Ruby kita mengenal istilah *symbol*. *Symbol* ini akan sering sekali anda temui di dalam Rails. Sering digunakan sebagai semacam konstanta pengganti string.

```
$irb> a = :test
$irb> a != :test
$=> false

$irb> a == :test
$=> true
```

Symbol ini bisa dibilang lebih hemat memori dibandingkan dengan String. Di dalam Ruby kita akan sering menemukan *symbol* dalam sebuah pemanggilan method. Selain itu kita juga akan sering menggunakan *symbol* sebagai sebuah *key* dalam *hash*. Kedua hal ini sering membuat bingung orang-orang yang pertama kali mempelajari Ruby.

² Interactive Ruby. Bisa digunakan untuk mempelajari Ruby tanpa harus membuat file .rb terlebih dahulu. Anda menjalankannya dengan mengeksekusi perintah *irb* pada console

2. Array dan Hash

Bagi anda yang sudah akrab dengan pemrograman pasti sudah mengenal apa yang dimaksud dengan *array* dan *hash*. Di dalam Ruby saya bisa mengatakan bahwa *array* sangat terkait dengan simbol `[]` sedangkan *hash* dengan `{}`. Mengapa? Karena memang itulah salah satu cara untuk menginisialisasi *array* dan *hash*.

```
$irb> a = []
$=> []

$irb> a = Array.new
$=> []

$irb> a = [47, 77, 17]
$=> [47, 77, 17]

$irb> a << 107
$=> [47, 77, 17, 107]

$irb> a.length
$=> 4

$irb> a.sort
$=> [17, 47, 77, 107]
```

Pada contoh diatas terlihat bahwa kita bisa juga melakukan inisialisasi *array* dengan memanggil method `new()`. Selanjutnya kita bisa menambahkan elemen pada *array* dengan menggunakan operator `<<`.

Berbeda dengan *array*, pada *hash* kita memiliki pasangan *key* dan *value*. Sebagaimana yang telah saya sebutkan sebelumnya, dalam Ruby *hash* ini seringkali dipakai sebagai parameter dalam pemanggilan sebuah method.

```
$irb> a = {}
$=> {}

$irb> a = Hash.new
$=> {}

$irb> a = {:keren => "abis"}
$=> {:keren => "abis"}

$irb> a[:manthab] = "jaya"
$=> "jaya"

$irb> a[:keren]
$=> "abis"

$irb> a[:manthab]
$=> "jaya"
```

Pada contoh di halaman sebelumnya terlihat bahwa kita bisa menggunakan *symbol* sebagai *key*. Namun, ternyata kita juga bisa melakukan hal yang sebaliknya sebagaimana yang terlihat pada contoh berikut ini.

```
$irb> a["keren"] = :jaya
$=> :jaya

$irb> a
$=> {:keren=>"abis", "keren"=>:jaya, :manthab=>"jaya"}
```

Sama seperti *array* kita bisa menggunakan method *length()* untuk melihat jumlah data yang ada di dalam *hash*.

```
$irb> a.length
$=> 3
```

3. Control Structure

Sekarang kita akan mempelajari *control structure* di Ruby. Sampai sini kita sebenarnya masih bisa menggunakan *irb*. Namun agar lebih jelas dan mudah dimengerti kita akan membuat sebuah file Ruby bernama coba1.rb.

```
a = ARGV[0].to_i

if a > 7
  puts "#{a} lebih besar dari 7"
elsif a < 7
  puts "#{a} lebih kecil dari 7"
else
  puts "#{a} sama dengan 7"
end
```

Perhatikan bahwa Ruby mengenali *elsif* bukan *else if*. Pada kode diatas kita melakukan substitusi nilai *a* di dalam tanda kutip dengan cara `"#{a}"`. Sekarang coba jalankan program di atas dan masukkan nilai sembarang sebagai argumen.

```
$ruby cobal.rb 8
$8 lebih besar dari 7
```

Kita akan sering menggunakan *looping* di dalam aplikasi kita. Salah satu caranya adalah dengan menggunakan *for*. Sekarang mari kita buat file baru coba2.rb yang di dalamnya kita isi dengan *code* berikut:

```
for i in 1..10 do
  puts "#{i}"
end
```

Ketika dijalankan, program ini akan mencetak angka 1 sampai dengan 10 ke layar.

Selain *for* kita juga bisa menggunakan *while* untuk melakukan *looping*. Berikut ini adalah contoh penggunaannya dalam file coba3.rb

```
i = 1
while i <= 10
  puts i
  i = i + 1
end
```

Ketika dieksekusi program ini akan memberikan output yang sama dengan coba2.rb.

4. Method

Sebuah *method* dideklarasikan dengan menggunakan *keyword* def dan diakhiri dengan end. Sebagai contoh buatlah file baru coba4.rb.

```
def coba_method(name)
  puts "coba #{name}"
end

a = "Jaya!"
coba_method(a)
coba_method("Dunks!")
```

Ketika dieksekusi, program diatas akan menghasilkan output sebagai berikut:

```
coba Jaya!
coba Dunks!
```

Untuk me-*return* sebuah nilai, kita hanya perlu menuliskan nilai (atau variable) tersebut di dalam method.

```
def kali_dua(nilai_awal)
  kali_dua = nilai_awal * 2
  kali_dua
end

kali_dua = kali_dua(4)
puts "kali dua #{kali_dua}"
```

Hasil eksekusi dari program ini adalah:

```
kali dua 8
```

Sebenarnya bisa saja kita menggunakan *keyword return* untuk mereturn sebuah nilai dari method. Namun, di sini saya hanya ingin menunjukkan alternatif dari cara tersebut.

5. Classes

Deklarasi sebuah kelas dimulai dengan *class* dan diakhiri dengan *end*. Sebagai contoh kita akan membuat sebuah kelas Mahasiswa pada file mahasiswa.rb.

```
class Mahasiswa
  def initialize(nama, npm)
    @nama = nama
    @npm = npm
  end
  def cetak_info
    puts "Nama: #{@nama}"
    puts "NPM: #{@npm}"
  end
end
```

Instance variable dari sebuah kelas dideklarasikan dengan '@' yang akan membuatnya bisa diakses oleh semua *instance method* yang ada di kelas tersebut. Namun *instance variable* tersebut tidak akan bisa diakses dari luar kelas. Untuk bisa melakukan itu kita perlu membuat *accessor method*. Tambahkan kedua *method* ini pada kelas Mahasiswa yang telah kita buat.

```
def nama=(nama_baru)
  @nama = nama_baru
end
def nama
  @nama
end
```

Anda bisa menguji method ini dengan melakukan pemanggilan method seperti dibawah ini.

```
mhs = Mahasiswa.new("Si Bocung", "1203000439")
mhs.cetak_info

mhs.nama = "Si Bocah" # mengubah nama menjadi Si Bocah
mhs.cetak_info

nama = mhs.nama # method ini akan mereturn Si Bocah sebagai nama
puts nama
```

Namun untuk mendefinisikan method *setter* dan *getter* bagi setiap variable adalah pekerjaan yang melelahkan. Oleh karena itu Ruby menyediakan method *attr_accessor()* untuk memudahkan kita.

```
class Mahasiswa
  attr_accessor :npm
  attr_accessor :nama
end
```


Selain *accessor* method ada lagi yang akan sering anda temukan di Rails, yaitu *class* method.

```
def self.whatami  
  puts "Saya Mahasiswa"  
end
```

Method ini diakses dengan diawali oleh nama kelasnya.

```
Mahasiswa.whatami # akan mencetak Saya Mahasiswa
```

Ada banyak hal lain mengenai *class* namun tidak bisa di bahas dalam tutorial ini seluruhnya. Anda bisa mempelajari hal-hal tersebut melalui link-link yang saya sediakan pada halaman selanjutnya.

6. Kemana Setelah Ini

Pada akhir tutorial ini anda telah mempelajari beberapa konsep di dalam bahasa Ruby. Saya tekankan sekali lagi bahwa tutorial ini hanyalah pengenalan Ruby sebelum anda mempelajari Rails. Namun, saya tidak akan melarang anda untuk mempelajari Ruby lebih lanjut setelah ini. Berikut ini adalah link website-website yang berguna untuk mempelajari Ruby.

1. **<http://poignantguide.net/ruby/>**
 - Belajar Ruby lewat novel dan komik.
2. **<http://tryruby.hobix.com>**
 - Tutorial Ruby yang sangat bagus. Anda bisa langsung mencobanya pada browser anda.
3. **<http://www.ruby-lang.org/>**
 - *Ruby: Programmers' Best Friend*. Ini adalah salah satu slogan dari website ini. Di sini anda bisa banyak hal mengenai Ruby termasuk link-link ke website Ruby yang lain.

Lampiran A: Menginstal Ruby Pada Windows

Tutorial ini akan menjelaskan langkah-langkah untuk menginstal Rails di komputer berbasis windows. Contoh yang akan diberikan dalam tutorial ini adalah cara menginstal Ruby on rails pada Windows 2000.

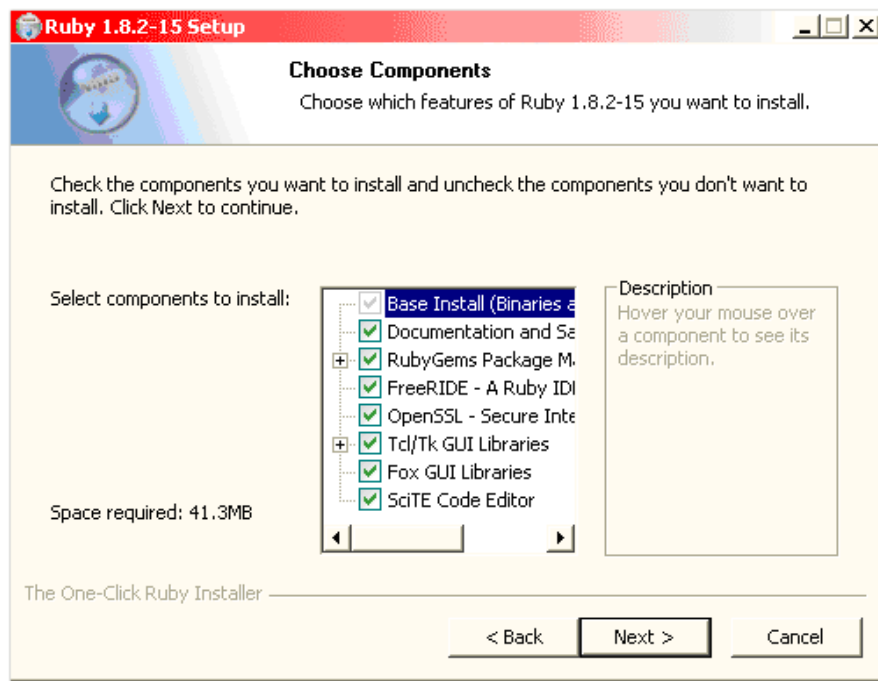
1. Download Ruby Installer for Windows

Anda bisa mendownload installer Ruby³ ini di <http://www.rubyforge.org>⁴

Versi terbaru dari ruby installer pada saat tutorial ini ditulis adalah Ruby1.8.2-15. Anda sangat disarankan untuk menggunakan versi terbaru yang stabil.

2. Jalankan Installer Ruby

Ikuti proses instalasi yang ada, anda mungkin akan memilih untuk tidak menginstall beberapa komponen yang ada jika memang anda tidak memerlukan. Tapi jika anda tidak tahu-menahu apa yang anda lakukan, dan anda tipe orang yang tidak memiliki masalah dengan *free space* di HardDisk anda, anda bisa menginstal semuanya.



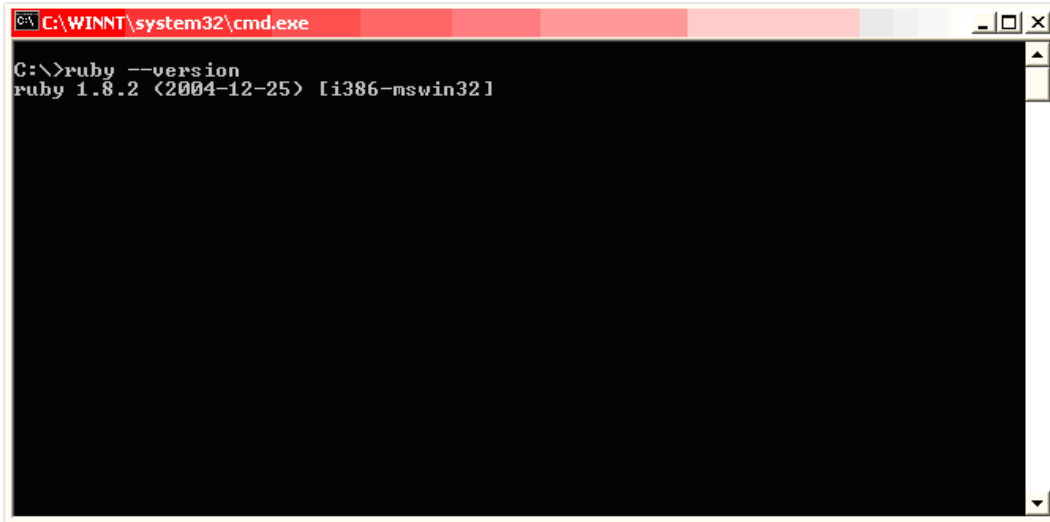
Gambar 1 - Proses Instalasi Ruby

³ Halaman project ruby installer untuk windows adalah <http://rubyforge.org/projects/rubyinstaller/>

⁴ <http://www.rubyforge.org> merupakan tempat dimana anda dapat mendownload distribusi Ruby dan juga extensionnya misalnya seperti driver untuk database PostgreSQL yang tidak diikut sertakan dalam installer Ruby.

3. Memastikan Ruby Telah Terinstal Dengan Sukses

Untuk memastikan bahwa Ruby telah terinstall dengan sukses dapat dilakukan dengan cara menjalankan perintah ruby --version pada konsol.



```
C:\WINNT\system32\cmd.exe
C:\>ruby --version
ruby 1.8.2 (2004-12-25) [i386-mswin32]
```

Gambar 2 - Melihat Versi Ruby

Jika anda berhasil, berarti anda telah sukses menginstall Ruby di Komputer anda.

Lampiran B: Biografi Penulis



Fajar Muharandy, mahasiswa Fakultas Ilmu Komputer UI angkatan 2003. Pernah menjabat sebagai ketua SIG Information System RiSTEK CSUI periode 2005-2006. Pada periode yang sama juga pernah menjadi kontributor *project runes* Enterprise Application Lab CSUI. Beberapa artikel dan tutorial yang ditulis di sini merupakan hasil kerjanya dalam *project runes*. Bidang minatnya adalah desain grafis, komputasi grid, dan E-Government. Sangat senang membaca dan menulis ^__^.

Feel free to contact me at:

e-mail : muharandy[at]yahoo[dot]com

Y!M : muharandy

Pengenalan Ruby on Rails

Fajar Muharandy

muharandy@yahoo.com

Lisensi Dokumen:

Copyright © 2003-2007 IlmuKomputer.Com

Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Pengembangan sebuah aplikasi web merupakan sesuatu yang sangat melelahkan jika kita harus membuat segala sesuatunya dari awal. Oleh karena itulah, sekarang ini banyak bermunculan *framework* yang selain mempercepat proses pengembangan, namun juga menawarkan suatu standar yang memungkinkan aplikasi web tersebut menjadi mudah untuk dikembangkan lebih lanjut.

Salah satu di antara *framework* yang ada tersebut adalah Rails yang dibuat menggunakan bahasa Ruby. Rails memungkinkan pengembangan aplikasi web yang tidak hanya cepat tapi juga rapi dan terstruktur dengan serangkaian standar dan aturan yang dimilikinya. Rails saat ini telah menjadi *framework* yang sedang naik daun dalam dunia pemrograman web. Bahkan, sang pembuat *framework* ini pun mengatakan bahwa impiannya adalah membuat Rails mampu bersaing dengan teknologi *mainstream* di dunia aplikasi *enterprise*, yaitu .NET dan Java.



Tentang Tutorial Ini

Tutorial ini merupakan pengenalan dengan framework Rails. Tutorial ini tidak bermaksud untuk membuat anda menjadi mahir menggunakan Rails dan langsung membuat aplikasi-aplikasi yang kompleks menggunakan Rails. Namun, tutorial ini lebih bertujuan sebagai langkah awal yang cepat dan mudah untuk mengenal Rails.

Siapa pun yang membaca tutorial ini disarankan membaca tutorial mengenai Ruby terlebih dahulu¹. Hal ini disebabkan karena akan banyak hal-hal yang berhubungan dengan Ruby yang akan dibicarakan di sini yang tidak akan dijelaskan secara detail.

Tutorial ini tidak akan terlalu banyak membicarakan konsep-konsep mengenai Rails, namun akan langsung mencoba menerapkan konsep tersebut dalam pembuatan sebuah aplikasi web sederhana. Namun dalam prosesnya, saya akan menjelaskan kepada anda mengenai hal-hal penting yang ada di dalamnya walaupun tidak akan menyeluruh.

Sebelumnya saya mohon maaf jika masih banyak terdapat kesalahan ataupun kekurangan di dalam tutorial ini karena saya sendiri juga masih dalam tahap belajar. Akhir kata, saya ingin mengucapkan terima kasih kepada siapa pun yang membaca tutorial ini, terutama bagi yang membaca halaman pertama yang penuh dengan hal-hal yang tidak penting ini. Semoga apa yang saya tulis di tutorial ini bisa bermanfaat bagi anda yang membacanya.

Penulis

¹ Ada tutorial pendamping yang saya tulis dengan judul “Pengenalan Pemrograman Ruby”. Tutorial pemrograman Ruby tersebut tidak menjelaskan pemrograman Ruby terlalu dalam, namun hanya sebagai referensi dari kode-kode Ruby yang saya gunakan dalam tutorial ini. Bagi anda yang ingin mempelajari Ruby lebih dulu, silahkan lihat website yang alamatnya saya berikan pada bab 4 tutorial ini.

Daftar Isi

Tentang Tutorial Ini.....	2
Daftar Isi	3
1. Pendahuluan.....	4
1.1 Rails Framework	4
1.2 Model, View, dan Controller	4
2. Membuat Proyek Baru.....	5
2.1 Membuat Direktori Proyek	5
2.2 Membuat Controller Sederhana	6
2.3 Pengaturan Database	9
3. Membuat Jurnal Sederhana	10
3.1 Rails Scaffold	10
3.2 Mengubah View.....	12
3.3 Membuat Model dan View, Serta Memodifikasi Controller	15
4. Kemana Setelah Ini	20
Lampiran A: Menginstal Ruby Pada Windows	21
Lampiran B: Menginstal Rails Pada Windows.....	23
Lampiran C: Biografi Penulis.....	26

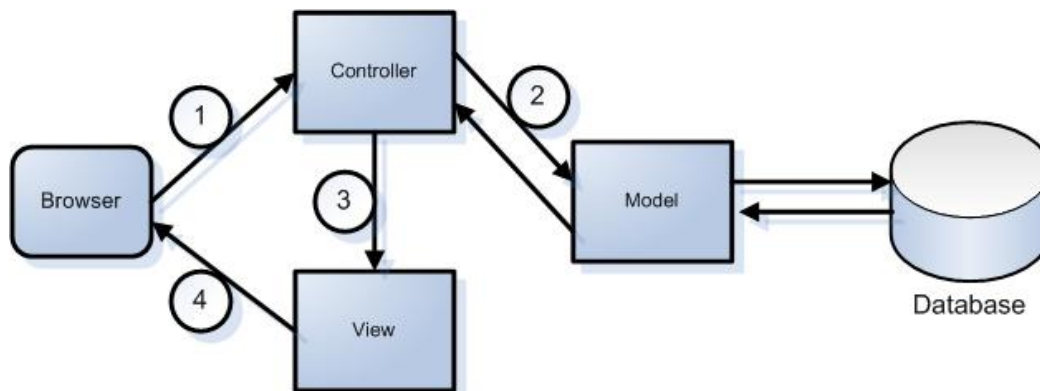
1. Pendahuluan

1.1 Rails Framework

Rails Merupakan sebuah framework MVC untuk pengembangan aplikasi berbasis web. Rails ditulis menggunakan bahasa pemrograman Ruby yang dikenal sangat *Object Oriented*. Di dalam pengembangan aplikasi menggunakan Rails kita akan mengenal sebuah paradigma baru yang dikenal dengan *convention over configuration*. Hal inilah yang membuat kita harus mengikuti segala sesuatu yang telah menjadi konvensi di dalam Rails. Misalnya, untuk penamaan tabel di database, Rails menentukan bahwa nama tabel haruslah kata benda jamak dari nama model yang kita miliki. Sebagai contoh, jika kita memiliki model Article maka kita harus memiliki tabel articles di database. Tetapi, hal ini bisa saja tidak kita ikuti dengan konsekuensi kita harus melakukan konfigurasi secara manual untuk melakukan *mapping* dari model Article ke table articles di database².

1.2 Model, View, dan Controller

Sebelumnya saya telah menyebut-nyebut nama MVC, namun sebenarnya apakah MVC itu?



Gambar 1 - Rails MVC Model

Saya lebih suka melihat model MVC dengan *controller* sebagai pusatnya karena hampir semua interaksi yang terjadi akan melalui *controller*. Bila dikaitkan dengan Rails, maka proses yang terjadi dalam model MVC secara umum adalah sebagai berikut³:

1. Browser akan mengirim *request* ke *controller*.
2. *Controller* akan meresponnya dan berkomunikasi dengan *model*. Komunikasi ini dapat berupa mengakses data, ataupun mengubah data yang disimpan oleh *model*. Perlu diperhatikan bahwa tidak semua *model* harus berhubungan dengan database.
3. *Controller* akan membuat *view* yang bersesuaian.
4. *Browser* akan me-render *view* yang ada.

² Hal ini tidak akan saya bahas di dalam tutorial ini. Bagi anda yang tertarik, anda bisa mencari tutorial mengenai *ActiveRecord* di Ruby yang sangat terkait dengan masalah ini.

³ Penjelasan ini saya ambil dari buku *Agile Web Development with Rails*, sehingga mungkin akan sedikit berbeda dengan konsep MVC apabila dikaitkan dengan framework lain.

2. Membuat Proyek Baru

2.1 Membuat Direktori Proyek

Hal yang pertama kali harus anda lakukan adalah membuat direktori proyek. Persiapan ini dapat dengan mudah anda lakukan karena Rails akan membuat dan mengaturnya secara otomatis untuk anda. Hal ini dapat dilakukan dengan menggunakan perintah *rails* pada direktori proyek yang anda inginkan.

```
$rails /path/nama_direktori_proyek
```

Perintah ini akan membuat direktori proyek dan hal-hal lain yang diperlukan. Untuk lebih jelasnya anda bisa melihat sendiri kedalam direktori proyek yang baru saja anda buat.

```
app/  
config/  
db/  
doc/  
lib/  
log/  
public/  
script/  
test/  
vendor/
```

Jika anda masuk ke dalam direktori *app/* maka anda akan melihat bahwa di dalamnya terdapat struktur MVC yang telah saya sebutkan sebelumnya, ditambah dengan direktori *helpers/*.

```
controllers/  
helpers/  
models/  
views/
```

Hal selanjutnya yang harus kita lakukan adalah menjalankan WebRick Server⁴. Pertama-tama, masuk ke dalam direktori *root* dari direktori proyek anda, kemudian jalankan perintah berikut ini.

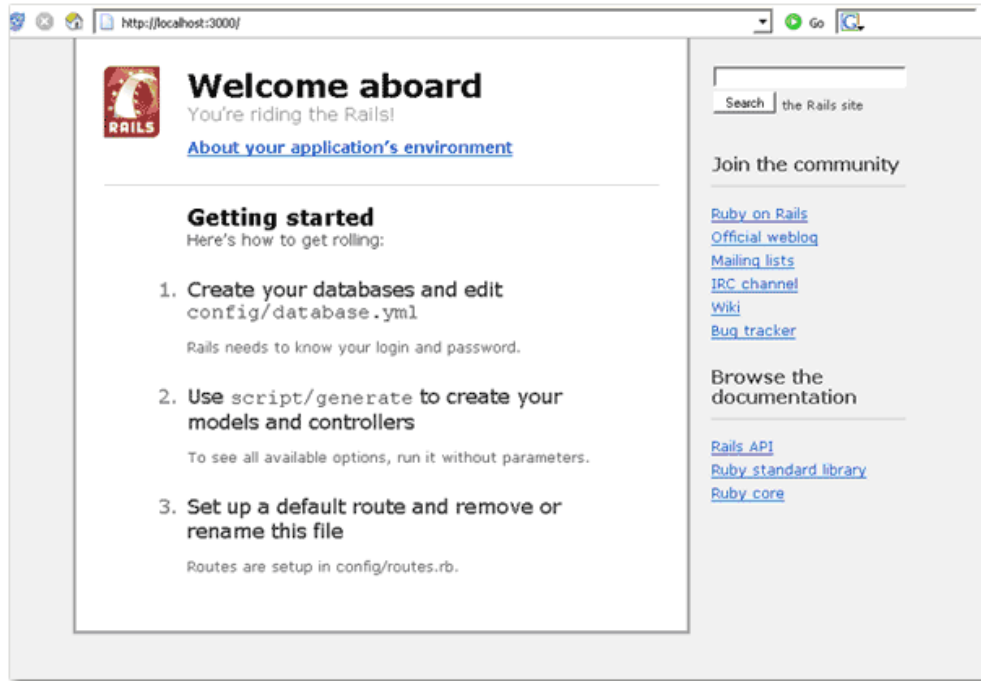
```
$ruby script/server
```

Anda akan melihat pesan-pesan yang menunjukkan bahwa WebRick telah dijalankan. Untuk memastikannya, anda dapat memeriksanya melalui *browser* anda. Masukkan alamat berikut pada *browser* anda

```
http://localhost:3000
```

⁴ Jika anda ingin men-*deploy* aplikasi anda pada lingkungan produksi sebaiknya anda tidak menggunakan WebRick. Anda bisa menggunakan *Apache*, *Lighttpd*, dll.

Jika benar maka anda akan melihat tampilan seperti pada **Gambar 2**⁵.



Gambar 2 - Tampilan Awal

2.2 Membuat *Controller* Sederhana

Seperti yang telah saya sebutkan sebelumnya, setiap *request* akan melalui *controller*. Untuk itu sekarang kita akan membuat sebuah *controller* sederhana yang akan memproses *request* dari *browser* dengan URL sebagai berikut:

`http://localhost:3000/Test/hehe`

Bagian pertama dari URL diatas merupakan alamat *root* direktori dari aplikasi kita, sedangkan yang selanjutnya yaitu Test, adalah nama dari *controller* kita. Kemudian yang terakhir adalah *action* dari *controller* yang ingin kita panggil. Di dalam Rails, secara umum kita dapat membaca URL tersebut dan mendaftar komponen-komponen yang terlibat sebagai berikut:

1. Sebuah kelas bernama *TestController*⁶
2. Sebuah method di dalam kelas *TestController* yang bernama *hehe*. Method dalam *controller* ini dikenal juga dengan sebutan *action*.
3. Sebuah file *.rhtml* yang berfungsi sebagai *view*⁷

⁵ Tampilan awal ini mungkin akan berbeda dengan yang anda miliki. Pada tutorial ini saya masih menggunakan Rails versi 1.1.2.

⁶ Ini akan otomatis dibuat ketika anda menjalankan perintah *ruby script/generate controller Test*. Anda tidak perlu membuat *controller* ini secara manual.

Anda tidak perlu membuat kelas dan *method* tersebut secara manual karena kita akan memanfaatkan perintah:

```
$ruby script/generate controller Test
```

Perintah ini akan secara otomatis membuat kita kelas `TestController` yang berada pada `app/controllers/test_controller.rb`⁸. Ketika anda membuka file tersebut anda akan menemukan sebuah implementasi kelas yang cukup sederhana.

```
class TestController < ApplicationController  
end
```

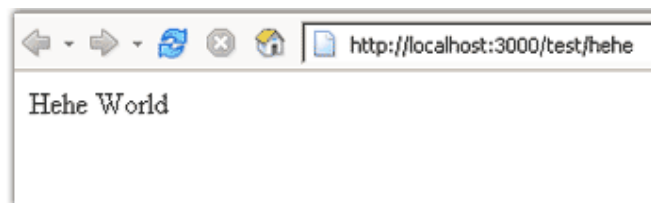
Selanjutnya anda perlu membuat method hehe karena kita ingin memiliki *action* hehe yang bisa kita akses melalui URL seperti yang sudah digambarkan sebelumnya.

```
class TestController < ApplicationController  
  def hehe  
    @hehe = "Hehe World"  
  end  
end
```

Mengingat konsep MVC yang telah saya sebutkan sebelumnya, maka kita memerlukan sebuah file yang bernama hehe.rhtml yang terletak pada `app/views/test`⁹. Buka editor favorit anda dan tulis ini di dalam file hehe.rhtml.

```
<html>  
  <body>  
    <%= @hehe %>  
  </body>  
</html>
```

Kemudian lihat hasilnya di *browser* anda pada <http://localhost/test/hehe>



Gambar 3 - Tampilan Pemanggilan Action hehe

⁷ Ingat konsep MVC yang telah diajarkan sebelumnya!

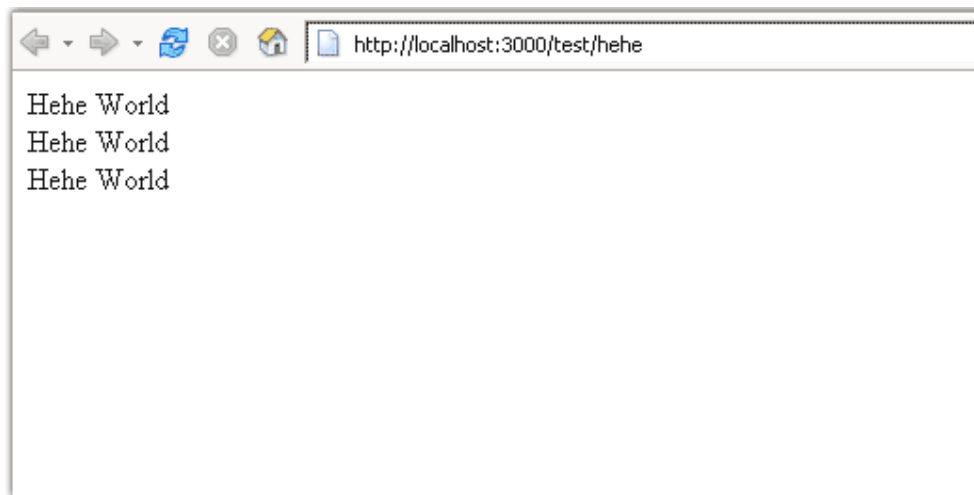
⁸ Jika anda memperhatikan, ini adalah apa yang sebelumnya saya sebut dengan *convention over configuration* di dalam Rails. Sebuah controller bernama *Test* akan diimplementasikan oleh kelas *TestController* yang terdapat di dalam file `test_controller.rb`

⁹ `app/views/test/hehe.rhtml`.... Anda dapat membacanya sebagai sebuah view untuk action bernama "hehe" dari controller "test"

Kode yang ditulis di dalam tag `<%= ... %>` akan diterjemahkan sebagai code dalam bahasa Ruby dan nilai yang dihasilkan oleh potongan kode tersebut akan ditulis untuk kemudian ditampilkan pada browser. Hal ini berbeda dengan tag `<% ... %>`. Dalam tag ini setiap code tetap akan diterjemahkan dalam bahasa Ruby namun hasil/output dari eksekusi potongan code tersebut tidak akan ditulis untuk ditampilkan ke *browser*. Sekarang kita akan merubah hehe.rhtml menjadi sebagai berikut:

```
<html>
  <body>
    <% i = 0 %>           → inisialisasi i
    <% while i < 3 %>     → while guard
      <%= @hehe %>       → cetak variabel @hehe
    <br />
    <% i = i + 1 %>      → increment i
    <% end %>
  </body>
</html>
```

Pada contoh yang terakhir ini saya melakukan inisialisasi variabel *i* di dalam tag `<% ... %>`. Begitu juga ketika saya mendefinisikan *condition* dari *while*, *increment* dari *i*, serta terminasi *while*. Saya baru mengganti nilai yang dihasilkan oleh kode dalam Ruby di atas (menggunakan tag `<%= .. %>`), pada saat saya ingin mencetak @hehe.



Gambar 4 - Tampilan Ketika Kita Memodifikasi hehe.rhtml

Anda bisa mencoba-coba hal lain yang telah anda ketahui dalam bahasa Ruby untuk ditampilkan pada hehe.rhtml.

2.3 Pengaturan Database

Aplikasi yang kite buat seringkali menggunakan DBMS sebagai sarana penyimpanan. Rails yang ditulis menggunakan Ruby tentu saja mendukung database yang juga telah didukung oleh Ruby. MySQL merupakan salah satu DBMS yang secara *default* didukung ketika anda menginstal Rails¹⁰. Pada tutorial ini saya akan menggunakan PostgreSQL¹¹ sebagai database, namun tidak berarti anda harus mengikutinya. Anda bisa menggunakan MySQL atau bahkan SQLite¹² sebagai langkah awal untuk belajar.

Langkah yang harus anda lakukan adalah membuka file konfigurasi database yang terletak di [config/database.yml](#)

```
development:
  adapter: postgresql
  database: trl
  username: radlspirit #isi dengan username anda
  password: sarivuyatsk #isi dengan password anda
  socket: localhost

test:
  ... ..

production:
  ... ..
```

Jika anda memperhatikan bagian bawah dari [database.yml](#) anda akan menemukan bahwa Rails menganjurkan anda untuk membuat dua database lain yang berbeda untuk fase *testing* dan *production*, sehingga total ada tiga database yang digunakan. Hal ini adalah *best-practice* dalam sebuah *software engineering* yang sebaiknya anda ikuti. Namun, dalam tutorial ini saya hanya akan menggunakan sebuah database saja untuk fase *development*¹³.

¹⁰ Hal ini menyebabkan anda harus menginstall *library* Ruby untuk membuat koneksi dengan DBMS selain MySQL

¹¹ The world's most advanced open source database! Yay!

¹² Database *embeddable* yang sangat-sangat *compact*.

¹³ *For the sake of simplicity and malesity!* Terkadang saya malas untuk melakukan hal-hal seperti ini, terutama ketika sedang belajar suatu hal yang baru.

3. Membuat Jurnal Sederhana

Pada bagian ini kita akan membuat sebuah jurnal sederhana dengan menggunakan Rails¹⁴. Kita akan menggunakan *code generator* dari Rails yang sebelumnya sudah pernah kita gunakan yaitu:

```
$ruby script/generate
```

3.1 Rails Scaffold

Namun sebelum kita mulai, kita harus membuat table di database terlebih dahulu. Karena kita akan membuat jurnal, maka kita akan membuat tabel dengan nama article di database. Berikut ini adalah DDL dari skema database yang saya buat di PostgreSQL:

```
create table articles (  
  id serial,  
  title varchar(50) not null,  
  time timestamp not null,  
  content text not null,  
  primary key(id)  
);
```

Sekarang saatnya memulai pertunjukan dengan Rails ^_^ . Kita akan meng-*generate model*, *view*, dan *controller* untuk jurnal kita ini dengan perintah *scaffold*. Karena ini adalah sebuah jurnal/blog sederhana maka saya akan membuat *controller* dengan nama Blog. Selain itu kita juga akan membuat model dengan nama Article.

```
$ruby script/generate scaffold Article Blog
```

Perlu diperhatikan di sini adalah hubungan antara tabel articles di database dengan *model Article*. Sebagaimana yang telah saya katakan sebelumnya, bahwa ini adalah konvensi yang ada di Rails. Sebuah tabel harus diberi nama dengan kata benda jamak dari *model* yang memilikinya. Dalam kasus ini kita memberi nama tabel articles untuk model bernama Article. Konvensi selanjutnya adalah, *primary key* dari tabel articles yang harus bernama id¹⁵. Selanjutnya buka browser anda dan masukkan alamat berikut ini:

```
http://localhost:3000/blog/
```

Alamat ini akan membuka list dari article yang telah anda buat dan disimpan di database. Namun karena saat ini anda belum menulis apa-apa, maka tampilannya kurang lebih akan seperti pada **Gambar 5**.

¹⁴ Jurnal ini benar-benar sebuah jurnal yang sangat sederhana. Jangan harap anda bisa menggantikan blog anda dengan ini. Walaupun anda bisa menggunakannya sebagai blog offline di komputer anda ^_^

¹⁵ Tidak bosan-bosannya saya mengatakan kalau konvensi ini bisa anda langgar dengan cara memodifikasi model anda secara manual. Anda bisa mempelajari tutorial tentang ActiveRecord di Ruby mengenai hal ini.



Gambar 5 - Listing articles

Sampai sini saya sarankan anda mencoba untuk membuat artikel baru, mengubah, dan menghapusnya dengan memanfaatkan *action-action* yang secara otomatis telah di-*generate* oleh Rails.

Selanjutnya buka file-file yang di-*generate* oleh perintah *scaffold* tadi. File-file penting yang harus diperhatikan adalah:

1. [app/controllers/blog_controller.rb](#): Di dalamnya terdapat implementasi kelas *BlogController*.
2. [app/models/article.rb](#): Di dalamnya terdapat implementasi kelas *Article* yang berfungsi sebagai *model*¹⁶.
3. [app/views/blog/list.rhtml](#): Ini merupakan *view* dari *action/method list* yang dimiliki oleh *BlogController*. Sebenarnya ada beberapa *view* lain yang dihasilkan, seperti [new.rhtml](#), [edit.rhtml](#), [show.rhtml](#). Perlu anda perhatikan bahwa semua *view* ini berkaitan dengan basic CRUD action yang disediakan oleh *controller*.

Perhatikan bagaimana method-method di kelas *kontroller* yang berfungsi sebagai *action* mempersiapkan data yang akan di-*render* oleh di *view* dengan cara mengakses *model article*¹⁷.

Saya tidak akan menjelaskan lebih lanjut mengenai method-method yang digunakan dalam *BlogController*, anda bisa membacanya sendiri dalam dokumentasi Rails di <http://api.rubyonrails.org> atau dengan menjalankan perintah *gem_server* dan kemudian membuka alamat <http://localhost:8808>¹⁸

¹⁶ Di sini anda dapat melihat betapa singkat dan sederhananya bahasa Ruby mengimplementasikan sebuah model

¹⁷ Inilah yang saya suka dari MVC, segala sesuatunya serba terstruktur dan rapi ^_^

¹⁸ Dokumentasi lokal ini bisa anda akses jika anda juga menginstall RDoc dari gem-gem yang ada.

3.2 Mengubah View

Tampilan blog yang baru saja kita buat sangatlah sederhana dan tidak menarik. Wajar saja, karena ini adalah tampilan *generic* dari Rails yang di-*generate* secara otomatis. Oleh karena itu, sekarang kita akan mencoba untuk mengubah *view* yang ada dengan cara mengaplikasikan *desain template* yang disertakan dalam modul tutorial ini¹⁹.

Pertama kita akan membuat *action* baru di kelas BlogController²⁰. *Action* ini akan diberi nama *index*. Dengan kata lain, kita harus mengimplementasikan method bernama *index* dalam BlogController, dan membuat *view* dengan nama *index.rhtml*.

Ketika membuka file *blog_controller.rb* kita akan dikejutkan dengan sudah adanya method *index* di dalamnya. Mengapa bisa seperti ini?! Apakah ini salah satu kehebatan Rails, yang mampu mengetahui segala sesuatu yang ada di benak kita?! Bahkan *action* yang “baru” saja akan kita buat, sudah langsung dibuatkan olehnya?! Wah.. Wah... Hebat! Hebaaat!²¹

Ubah implementasi method *index* menjadi seperti dibawah ini:

```
def index
  @articles = Article.find(:all, :order => "time desc")
end
```

Method find di dalam kelas article akan me-*return* artikel-artikel yang ada di dalam database. *:all* menunjukkan bahwa kita ingin semua artikel yang ada agar di-*return*. Opsi lain adalah *:first* yang akan mereturn *record* pertama yang memenuhi kondisi yang telah kita tentukan.

Di sini saya tidak menggunakan teknik *paging* sebagaimana yang digunakan dalam method *list*. Artikel yang diambil akan diurutkan berdasarkan waktu dari yang terbaru ke yang paling lama dengan parameter hash *:order => "time desc"*.

Selanjutnya kita membutuhkan *view* untuk *action* *index*. Namun, sebelum kita membuat *index.rhtml* di dalam direktori *app/views/blog*. Kita akan memanfaatkan *design template* yang terdapat di dalam *casciscusblah.zip*.

1. Ekstrak file ini, anda bisa melihat *mock-up interface* dari aplikasi anda pada file *index.html* dan *comment.html*.
2. Kemudian kopi file *style.css* yang ada di dalam direktori *stylesheets* ke dalam direktori *public/stylesheets* di direktori proyek anda.
3. Lakukan hal yang sama terhadap semua file yang ada di dalam direktori *images*. Kopi semuanya ke dalam direktori *public/images* yang ada di dalam direktori proyek anda.

¹⁹ casciscusblah.zip berisi contoh desain dari blog sederhana yang dibuat khusus untuk tutorial ini.

²⁰ Kelas BlogController ini terletak pada *app/controllers/blog_controller.rb*

²¹ Kenyataannya tidak seperti itu... Saya hanya sedang bingung mengenai apa yang harus saya tulis selanjutnya... Method ini adalah *default* dari perintah scaffold dan isinya adalah memanggil *action list*

Setelah melakukan itu semua, anda perlu mendefinisikan Layout baru yang akan dijadikan semacam *design template* oleh *view* yang anda buat nantinya. Caranya adalah dengan mengubah file *blog.rhtml* yang ada di dalam *app/views/layouts*. File ini merupakan layout dari *controller* Blog yang telah anda buat sebelumnya. Ubah isinya menjadi seperti berikut:

```
1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
3. <head>
4. <title>CasCisCusBlah</title>
5. <!-- meload style.css -->
6. <%= stylesheet_link_tag "style", :media => "all" %>
7. </head>
8.
9. <body id="casciscus">
10. <div id="container">
11.   <div id="header">
12.     <h1><span>CasCisCusBlah.Blog</span></h1>
13.   </div>
14.   <div id="content">
15.     <p style="color: green">
16.       <!-- menampilkan pesan yang di-generate controller -->
17.       <%= flash[:notice] %>
18.     </p>
19.     <!-- meload content -->
20.     <%= @content_for_layout %>
21.   </div>
22.   <div id="footer">
23.     <p>Copyright &copy; radlStudio, 2006. All Rights Reserved</p>
24.     <br>
25.   </div>
26. </div>
27. </body>
28. </html>
```

Pada contoh diatas kita menggunakan *helper* method *stylesheet_link_tag()* untuk me-load *style.css*. Pada baris ke-13 kita menampilkan pesan yang dikirim dari *controller* yang disimpan dalam *Flash*. Selanjutnya pada baris ke-14 kita meload content yang ada. Pada baris inilah output yang dihasilkan oleh sebuah *view* dari *action* pada *BlogController* akan ditempatkan.

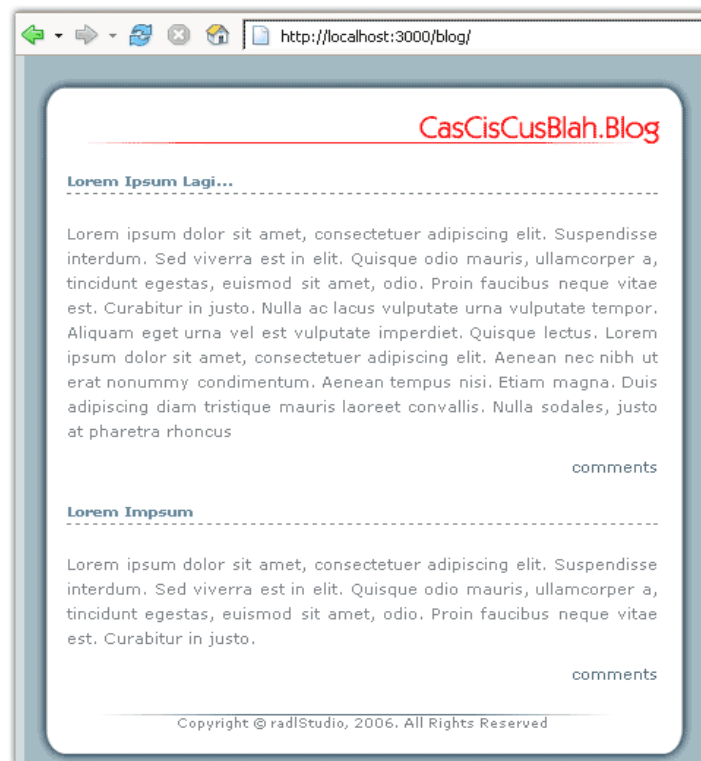
Kita belum berhenti sampai di sini karena kita masih harus membuat file *index.rhtml* untuk menampilkan halaman *index*. Oleh karena itu, kita buat file baru *index.rhtml* dalam direktori *app/views/blog*. Anda kemudian bisa mengisinya dengan potongan *code* HTML yang terdapat

di dalam file [index.html](#) dari [casciscusblah.zip](#). Anda tidak perlu memasukkan bagian header dan footer dari file tersebut, karena bagian tersebut telah kita definisikan pada [layouts/blog.rhtml](#). Ini adalah salah satu kegunaan menggunakan *layout*.

Masukkan beberapa baris code ini pada [index.rhtml](#) yang baru saja anda buat.

```
1. <% for article in @articles %>
2. <h2><%= article.title %></h2>
3. <p>
4. <%= article.content %>
5. </p>
6. <div id="bottom_link">
7.   <a href="">comments</a>
8. </div>
9. <% end %>
```

Ingat bahwa tadi pada BlogController kita men-set *field* [@articles](#) dengan semua artikel-artikel yang ada di dalam database. Kita akan melakukan iterasi pada [@articles](#) dan mengambil *title* serta *content* dari masing-masing artikel. Untuk sementara kita abaikan dulu link *comments* yang ada. Hasil dari pemanggilan *action* [index](#) yang telah kita buat adalah sebagai berikut²².



Gambar 6 - Tampilan halaman index

²² Pastikan anda telah membuat artikel-artikel *dummy* untuk ditampilkan

Jika anda mencoba memanggil *action* *list* dengan cara menuliskan alamat *http://localhost:3000/blog/list*, maka anda akan di hadapkan dengan tampilan yang sedikit berantakan. Hal ini disebabkan karena *view* pada *list.rhtml* mencoba untuk menampilkan semua isi content dari sebuah artikel. Oleh karena itu kita modifikasi *list.rhtml* tepatnya pada baris ke-13 sehingga setiap string yang ditampilkan akan di-*truncate*.

```
13. <%=h truncate(article.send(column.name).to_s, 50)
```

Sekarang coba *refresh browser* anda, dan anda akan mendapatkan tampilan yang lebih rapi dibandingkan sebelumnya.



Gambar 7 - Tampilan *list.rhtml* yang telah dimodifikasi

Sampai sini anda telah berhasil memodifikasi *view* dari aplikasi kita. Tidak hanya itu, anda juga telah melakukan modifikasi pada *action* *index* untuk mengambil data *article* dari database melalui *model* yang ada. Namun, kita tidak akan berhenti sampai di sini, karena kita masih harus membuat fasilitas untuk menambahkan komentar pada jurnal kita.

3.3 Membuat Model dan View, Serta Memodifikasi Controller

Sebelumnya kita selalu memanfaatkan kode yang telah di-*generate* secara otomatis oleh Rails melalui perintah *scaffold*. Kali ini kita akan membuat sebuah fungsi baru untuk menambahkan komentar pada jurnal kita. Dengan kata lain, kita memerlukan *model*, *view*, dan *action* baru pada *controller* untuk menambahkan fungsi tersebut.

Pertama-tama yang harus kita lakukan adalah membuat tabel *comments* di dalam database.

```
create table comments (  
  id serial,  
  article_id int references articles(id),  
  name varchar(50) not null,  
  content text not null,  
  primary key(id)  
);
```

Perhatikan bahwa kita harus mendefinisikan *foreign key* article_id yang me-refer pada id di table articles. Ini merupakan salah satu konvensi lagi di dalam Rails, dimana *foreign_key* diberi nama dengan ketentuan nama tabel (singular) dimana dia me-refer, diikuti dengan '_' dan suffix *id*. Contohnya tabel *comments* yang me-refer kepada tabel *articles* akan memiliki *article_id* sebagai *foreign key*.

Setelah mempersiapkan tabelnya, sekarang kita harus meng-*generate model* dari Comment.

```
$ruby script/generate model Comment
```

Kita bisa melihat hasil dari perintah ini di direktori app/models/comment.rb. Sekali lagi kita akan menemukan sebuah implementasi sederhana yang hanya terdiri dari dua baris kode. Namun, masih ada yang kurang dari implementasi *model* Comment ini. Kita belum mendefinisikan bahwa *model* Comment ini berelasi dengan *model* Article. Oleh karena itu kita tambahkan satu baris pemanggilan method *belongs_to()* pada kelas Comment untuk memberitahu kepada Rails mengenai hubungan antara Comment ini dengan Article.

```
class Comment < ActiveRecord::Base  
  belongs_to :article  
end
```

Kita juga harus melakukan hal sama pada *model* Article. Buka implementasi kelas Article yang terletak pada app/models/article.rb. kemudian tambahkan pemanggilan method *has_many()*²³.

```
class Article < ActiveRecord::Base  
  has_many :comments  
end
```

Kita telah mempersiapkan *model* yang kita butuhkan untuk fungsi baru ini. Tapi kita masih perlu menambahkan *actions* dan *view* untuk fungsi ini. Sekarang buka Kelas BlogController di dalam app/controllers/blog_controller.rb dan tambahkan implementasi *comment()*.

```
def comment  
  @article = Article.find(params[:id])  
  @comment = Comment.new  
end
```

²³ Saya asumsikan anda memiliki pengetahuan yang cukup tentang relasi antar *entity* dalam database.

Kita lihat bahwa pertama-tama kita mengambil data dari sebuah artikel yang ingin kita beri komentar dengan *method find()*. Selanjutnya kita membuat objek *Comment* yang akan melakukan *binding* terhadap data-data yang di-*input* oleh user.

Selanjutnya kita akan melakukan modifikasi pada *index.rhtml* sehingga *action comment* ini akan dipanggil ketika user mengklik link *comments*. Hal ini bisa kita peroleh dengan cara melakukan pemanggilan terhadap *method link_to()*²⁴ seperti yang terlihat pada baris ke-7 pada kode dibawah ini.

```
1. <% for article in @articles %>
2. <h2><%= article.title %></h2>
3. <p>
4.   <%= article.content %>
5. </p>
6. <div id="bottom_link">
7.   <%= link_to 'comments', :action => 'comment', :id => article %>
8. </div>
9. <% end %>
```

Hmm, setelah *action comment* ini dipanggil, apa selanjutnya? Tentu saja Rails akan merender *view* dari action ini²⁵. Tapi, kita belum membuat *view* yang dimaksud. Oleh karena itu buatlah sebuah file baru bernama *comment.rhtml*.

```
1. <h2><%= @article.title %></h2>
2. <p>
3.   <%= @article.content %>
4. </p>
5. <div id="comment">
6.   <% for comment in @article.comments %>
7.     <hr>
8.     <b><%= comment.name %></b>
9.     <p>
10.       <%= comment.content %>
11.     </p>
12.   <% end %>
13. </div>
14. <hr>
15. <div id="input">
16. <%= start_form_tag(:action => "save_comment", :id => @article)%>
17.   <table>
18.     <tr>
19.       <td>name</td>
20.       <td><%= text_field("comment","name", "size" => 20) %></td>
```

²⁴ Anda akan menemukan bahwa *article.id* akan sama dengan *article*. Seperti yang saya gunakan pada parameter terakhir dari *hash options* method *link_to(..., ..., :article => article)*.

²⁵ Sekali lagi ^^ inilah yang saya suka dari MVC. Strukturnya membantu saya untuk mengingat apa yang harus saya buat selanjutnya. Hohoho, mari kita buat segalanya menjadi lebih rapi dan terstruktur.

```
21.         </tr>
22.         <tr>
23.             <td>comment</td>
24.             <td><%= text_area("comment","content","rows" => 5, "cols" =>
25.                 50) %></td>
26.         </tr>
27.         <td></td>
28.         <td style="text-align: right">
29.             <input class="submit_button" type="submit" name="submit"
30.                 value="Submit">
31.         </td>
32.     </tr>
33. </table>
34. <%= end_form_tag %>
35. </div>
```

Tidak usah bingung melihat kode dalam file ini. Intinya, *view* ini terdiri dari tiga bagian. Pertama adalah bagian dimana kita menampilkan kembali artikel yang akan diberi komentar (baris ke-1 s/d 4). Yang kedua adalah bagian yang menampilkan komentar yang sudah masuk (baris ke-5 s/d 13). Terakhir adalah bagian yang menampilkan *form* untuk menambahkan komentar (baris ke-15 s/d 34).

Di awal *form* kita memanggil method *start_form_tag()*. Melalui pemanggilan method ini kita telah mendefinisikan bahwa *action* yang akan dipanggil adalah *save_comment*. Kita juga mengirimkan *id* dari artikel yang akan diberi komentar ke *action* ini melalui parameter *hash options :id*.

Selanjutnya untuk membuat input *text field* dimana pengguna bisa memasukkan namanya, kita memanfaatkan method *text_field()*. Dua parameter pertama dari *method* ini secara berurutan adalah nama objek dan *attribut/field* dari *model* yang sebelumnya kita buat pada pemanggilan method *comment()*. Dengan ini Rails melakukan *binding* terhadap data yang dimasukkan ke dalam field tersebut dengan *model* yang kita miliki. Hal yang sama kita lakukan terhadap input *text area*.

Sekarang kita perlu mendefinisikan *method* baru di dalam kelas *BlogController* yang akan berfungsi sebagai *action* untuk menyimpan komentar.

```
def save_comment
  @comment = Comment.new(params[:comment])
  @article = Article.find(params[:id])
  @article.comments << @comment
  @article.save

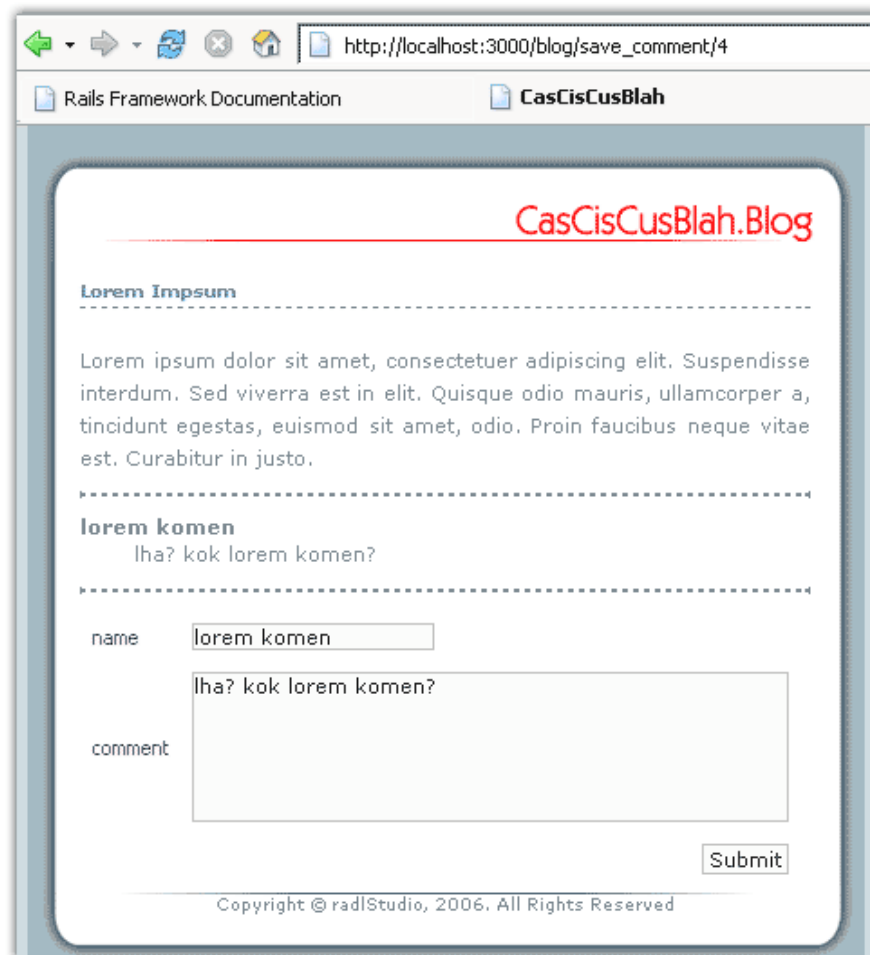
  render(:action => 'comment')
end
```

Perhatikan bagaimana kita menambahkan komentar pada suatu artikel dengan menggunakan operator << dan kemudian menyimpannya.

```
@article.comments << @comment
@article.save
```

Mengapa *article* yang kita simpan? Kita bisa melakukan penyimpanan seperti itu karena kita telah mendefinisikan bahwa *Comment belongs_to Article* dan *Article has_many Comments*. Kalau anda jeli melihat [*comment.rhtml*](#) tepatnya pada baris ke-6. Anda akan melihat bahwa kita bisa mengakses komentar-komentar yang dimiliki oleh sebuah *Article* melalui *@article.comments*. Wow!!

Terakhir kita melakukan pemanggilan terhadap *action comment* untuk menampilkan kembali halaman komentar beserta komentar yang baru dimasukkan²⁶.



Gambar 8 - Lorem Komen

²⁶ Seperti saat membuat artikel, saya tidak mencontohkan cara melakukan validasi dari data-data yang dimasukkan user. Sekali lagi.. *For the sake of simplicity and malesity* ^_^

4. Kemana Setelah Ini

Pada akhir tutorial ini anda telah mulai mempelajari dasar-dasar Rails. Namun, sebagaimana yang telah saya sebutkan sebelumnya, tutorial ini hanya bertujuan untuk memperkenalkan Rails kepada anda dan tidak bermaksud untuk membuat anda menjadi mahir menggunakan Rails. Meskipun demikian, proses pembelajaran tidak akan berhenti sampai di sini. Ada banyak sekali sumber-sumber yang bisa anda jadikan acuan untuk mempelajari dan menggunakan Rails. Berikut ini adalah beberapa diantaranya:

1. [**http://poignantguide.net/ruby/**](http://poignantguide.net/ruby/)
 - Belajar Ruby lewat novel dan komik
2. [**http://tryruby.hobix.com/**](http://tryruby.hobix.com/)
 - Mencoba pemrograman Ruby secara interaktif di browser anda.
3. [**http://www.rubyonrails.org**](http://www.rubyonrails.org)
 - Website resmi dari Rails framework.
4. [**http://api.rubyonrails.org**](http://api.rubyonrails.org)
 - Bagi anda yang memperhatikan **Gambar 8** anda akan melihat bahwa saya sedang membuka dokumentasi Rails Framework di website ini.
5. [**http://www.rubyforge.org**](http://www.rubyforge.org)
 - Tempat mencari *update* terbaru dari Ruby dan gem-gemnya.
6. [**http://www.ilmukomputer.com**](http://www.ilmukomputer.com)
 - Kemana lagi selain ke website di mana anda mendapatkan tutorial ini. Siapa tahu saya menulis *update* terbaru untuk tutorial ini.

Lampiran A: Menginstal Ruby Pada Windows

Tutorial ini akan menjelaskan langkah-langkah untuk menginstal Rails di komputer berbasis windows. Contoh yang akan diberikan dalam tutorial ini adalah cara menginstal Ruby on rails pada Windows 2000.

Sebelum mencoba kedahsyatan Rails Framework anda terlebih dahulu harus menginstal Ruby, karena Rails tidak akan berjalan tanpa Ruby interpreter.

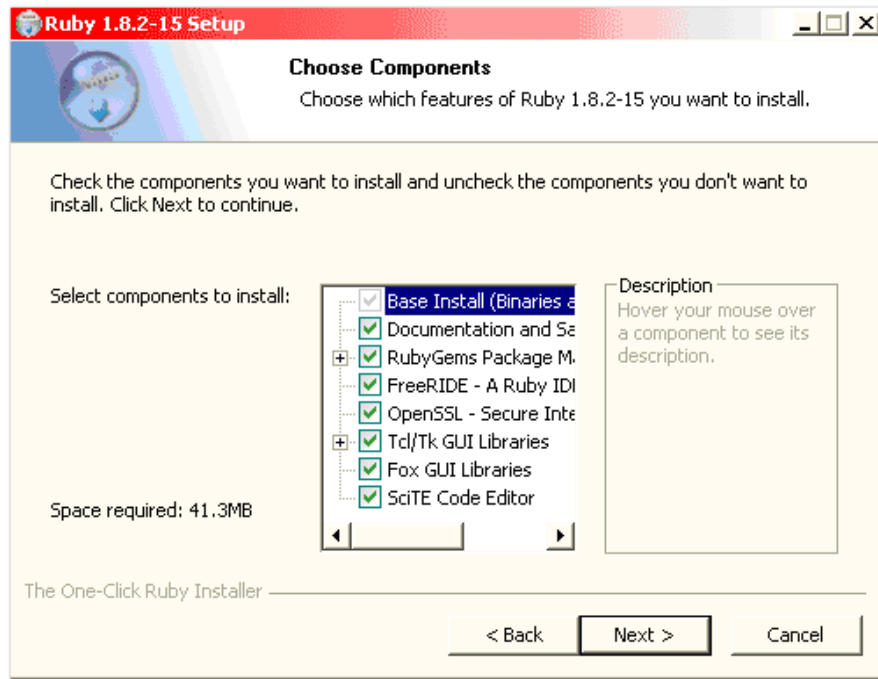
1. Download Ruby Installer for Windows

Anda bisa mendownload installer Ruby²⁷ ini di [²⁸http://www.rubyforge.org](http://www.rubyforge.org)

Versi terbaru dari ruby installer pada saat tutorial ini ditulis adalah Ruby1.8.2-15. Anda sangat disarankan untuk menggunakan versi terbaru yang stabil.

2. Jalankan Installer Ruby

Ikuti proses instalasi yang ada, anda mungkin akan memilih untuk tidak menginstall beberapa komponen yang ada jika memang anda tidak memerlukan. Tapi jika anda tidak tahu-menahu apa yang anda lakukan, dan anda tipe orang yang tidak memiliki masalah dengan *free space* di HardDisk anda, anda bisa menginstal semuanya.



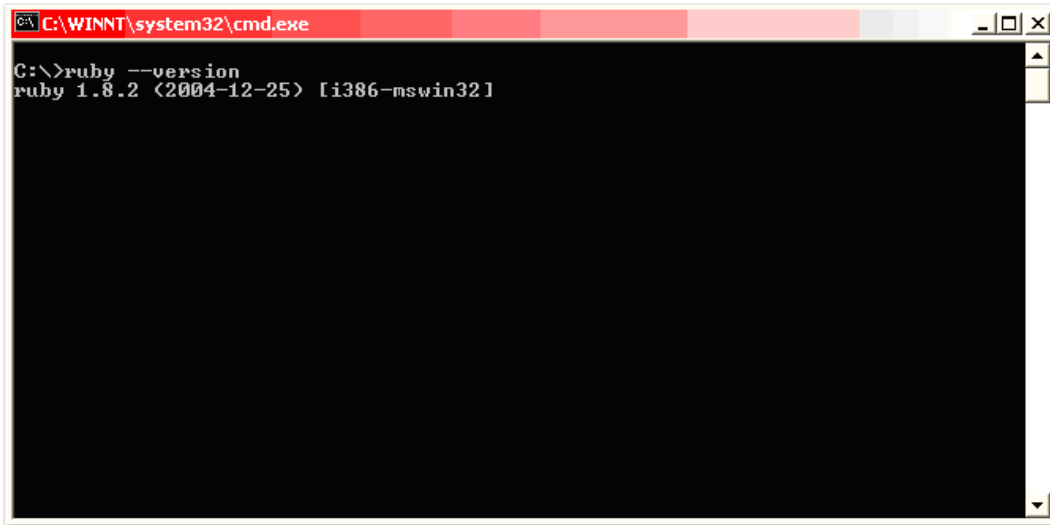
Gambar 9 - Proses Instalasi Ruby

²⁷ Halaman project ruby installer untuk windows adalah <http://rubyforge.org/projects/rubyinstaller/>

²⁸ <http://www.rubyforge.org> merupakan tempat dimana anda dapat mendownload distribusi Ruby dan juga extensionnya misalnya seperti driver untuk database PostgreSQL yang tidak diikut sertakan dalam installer Ruby.

3. Memastikan Ruby Telah Terinstal Dengan Sukses

Untuk memastikan bahwa Ruby telah terinstall dengan sukses dapat dilakukan dengan cara menjalankan perintah ruby --version pada konsol.



```
C:\WINNT\system32\cmd.exe
C:\>ruby --version
ruby 1.8.2 (2004-12-25) [i386-mswin32]
```

Gambar 10 - Melihat Versi Ruby

Jika anda berhasil, berarti anda telah sukses menginstall Ruby di Komputer anda²⁹.

²⁹ Tahap selanjutnya setelah ini adalah terserah anda. Apakah anda ingin langsung bermain-main dengan Ruby, atau lanjut untuk menginstall Rails Framework. Tidak ada unsur paksaan dalam tutorial ini...

Lampiran B: Menginstal Rails Pada Windows

Proses instalasi Rails yang akan dijelaskan disini tidak mencakup bagaimana mengintegrasikannya dengan webserver yang sudah ada seperti Apache³⁰. Untuk proses development dan pembelajaran, Rails telah mengikutsertakan WEBrick³¹.

1. Menginstal Rails beserta dependencies menggunakan *gem*

Pada distribusi Ruby installer, secara *default* anda akan mendapatkan *gem*³² terinstal secara otomatis. Kemudian, jalankan perintah ini pada console:

```
gem install rails --include-dependencies
```

Perintah ini akan menginstall rails beserta paket-paket yang menjadi prasyaratnya. Namun, entah kenapa di Windows 2000 yang saya gunakan, perintah ini tidak berjalan sebagaimana mestinya. Saya belum pernah mencobanya pada WinXP, namun jika anda mengalami hal yang sama maka anda harus menggunakan cara manual seperti yang akan saya jelaskan selanjutnya.

2. Instal Rails beserta dependencies satu per satu menggunakan *gem*

Anda terlebih dahulu harus mengetahui dependency dari rails, yaitu:

```
rake (>= 0.6.2)
activesupport (= 1.2.5)
activerecord (= 1.13.2)
actionpack (= 1.11.2)
actionmailer (= 1.1.5)
actionwebservice (= 1.0.0)
```

List diatas saya peroleh dengan menggunakan perintah:

```
gem dependency rails
```

Jangan panik jika anda tidak bisa memunculkan list dependencies-nya... toh saya telah memberikannya untuk anda :P. Prasyarat versi dari masing-masing paket mungkin tidak akan sama dengan yang saya tampilkan di sini.

Selanjutnya satu per satu modul tersebut anda install dengan menggunakan perintah:

```
gem install nama_modul
```

dan diakhiri dengan menginstall Rails itu sendiri

```
gem install rails
```

³⁰ Untuk mengintegrasikan aplikasi rails anda agar dapat berjalan diatas Apache anda dapat bertanya pada mbah google

³¹ WEBrick adalah server yang ditulis dalam bahasa Ruby. Penggunaannya pun sangat mudah dan sangat berguna dalam proses development. Namun untuk deployment aplikasi yang sebenarnya disarankan menggunakan webserver lain yang lebih handal

³² Bagi yang sudah pernah menggunakan debian dan merasakan indahnya dunia linux dengan apt-get, mungkin akan mengenali gem sbg apt-get versi Ruby.

Dengan ini anda telah sukses menginstall Rails pada komputer anda dan siap bersenang-senang dengannya.

3. Memastikan instalasi rails

Untuk memastikan bahwa anda telah sukses menginstall Rails, caranya adalah dengan membuat proyek sederhana menggunakan Rails itu sendiri. Pertama-tama buat sebuah direktori yang akan anda jadikan tempat project Rails anda yang baru, contoh³³:

```
C:\radlspirit\project\rails\myfirstrails
```

Kemudian jalankan perintah `rails` di console yang akan generate file-file dan struktur direktori yang diperlukan dalam proses development, contoh:

```
rails C:\radlspirit\project\rails\myfirstrails
```

Anda bisa melihat struktur direktori dan file-file yang di-generate pada direktori project anda. Setelah puas melihat-lihat, anda dapat lanjut ke langkah selanjutnya.

4. Menjalankan aplikasi Rails yang baru anda buat

Anda memang belum membuat apa-apa, namun tidak ada salahnya anda mengikuti beberapa langkah berikut...

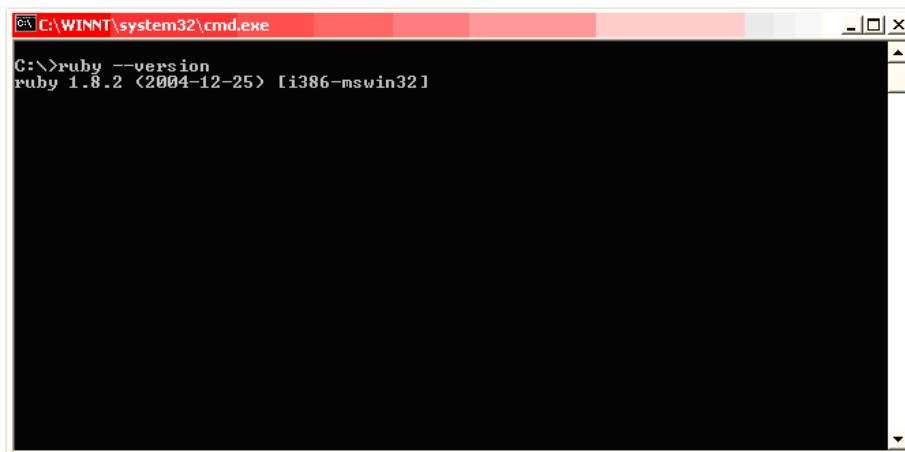
Lewat konsol, masuk ke direktori proyek anda:

```
cd C:\radlspirit\project\rails\myfirstrails
```

Kemudian lakukan perintah ini untuk menjalankan WEBrick:

```
ruby script/server
```

Perintah ini akan menjalankan WEBrick pada port 3000. Berikut ini adalah tampilan pada konsol ketika WEBrick sukses dijalankan.



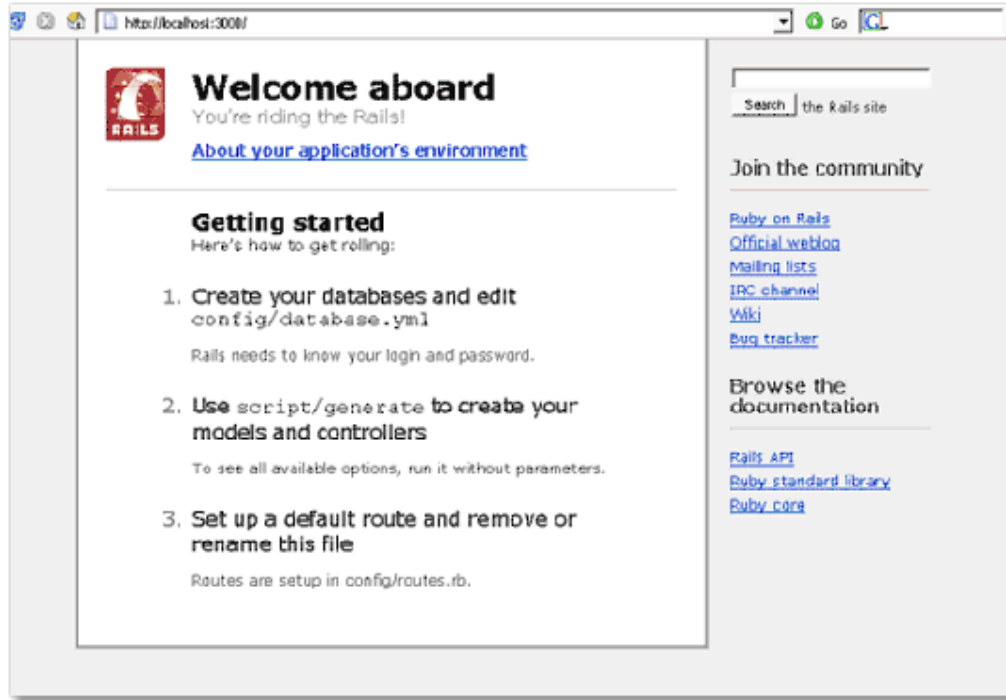
Gambar 11 - Menjalankan WEBrick

³³ Ini hanya contoh saja, anda tidak harus ikut-ikutan membuat direktori seperti ini. Tidak ada unsur paksaan dalam tutorial ini...

Selanjutnya buka browser favorit anda dan masukkan alamat ini pada address:

<http://localhost:3000>

Mungkin hasilnya akan sedikit berbeda tergantung versi Rails yang anda gunakan.



Gambar 12 - Tampilan Awal Rails

Lampiran C: Biografi Penulis



Fajar Muharandy, mahasiswa Fakultas Ilmu Komputer UI angkatan 2003. Pernah menjabat sebagai ketua SIG Information System RiSTEK CSUI periode 2005-2006. Pada periode yang sama juga pernah menjadi kontributor *project runes* Enterprise Application Lab CSUI. Beberapa artikel dan tutorial yang ditulis di sini merupakan hasil kerjanya dalam *project runes* Bidang minatnya adalah desain grafis, komputasi grid, dan E-Government. Sangat senang membaca dan menulis ^__^.

Feel free to contact me at:

e-mail : muharandy[at]yahoo[dot]com

Y!M : muharandy