Sesi 4: Libraries & Testing

Libraries

Salah satu keunggulan ruby adalah ruby memiliki banyak sekali library yang bisa kita pakai sesuai kebutuhan kita. Standar distribusi library di ruby adalah dengan menggunakan format yang bernama *gem*.

Instalasi & menggunakan gems

Sebuah gem adalah suatu kumpulan file library beserta metadatanya yang dikompresi dengan gzip, biasanya dengan ekstensi `.gem`. Untuk menginstall suatu library menggunakan gem, kita menggunakan perintah

```
  $ gem install <nama gem>
```

contoh, kita akan menginstall builder, library untuk membuat dokumen xml:

```
  $ gem install builder
```

Untuk menggunakan library, baik yang bawaan ruby maupun yang diinstall menggunakan gem, kita menggunakan method `require "<nama file>"`.

```
require "rubygems" # tidak dibutuhkan untuk ruby 1.9 ke atas
require "builder"

xml = Builder::XmlMarkup.new
xml.person do
  xml.name "Slamet Riyadi"
  xml.location "Indonesia"
end
```

Bundler

Untuk memanage library-library yang digunakan pada suatu project, kita dapat menggunakan bundler.

Pada root directory suatu project, kita mencatat semua library yang kita gunakan beserta versi masing-masing pada file dengan nama *Gemfile*.

```
mkdir my_project
cd my_project
touch Gemfile

# Gemfile
source 'http://rubygems.org'

gem 'rails', '3.2.3'
gem "authlogic"
gem 'will_paginate', :git => "git://github.com/bukalapak/will_paginate.git"
gem "simple_form", "~> 2.0"
gem "whenever", :require => false
```

Untuk menginstall library yang dibutuhkan, kita menggunakan perintah `bundle install`. Bundler kemudian akan membuat file dengan nama *Gemfile.lock* yang mencatat versi library yang digunakan saat ini, sehingga ketika kita menjalankan `bundle install` pada lain waktu, kita akan mendapatkan library dengan versi yang sama.

Kemudian pada project kita, kita me-load library menggunakan bundler sebagai berikut:

```
require "rubygems"
require "bundler/setup"

require "authlogic"
```

Packaging Programs and Libraries for Distribution

Untuk membuat kode kita menjadi gem, kita akan menggunakan bundler untuk menyiapkan package dan strukturnya, sebagai berikut:

```
bundle gem namagem
```

yang akan menghasilkan struktur direktori di mana kita bisa meletakkan file-file library kita. Kemudian kita edit file namagem.gemspec dengan data yang sesuai:

```
# -*- encoding: utf-8 -*-
$:.push File.expand_path("../lib", __FILE__)
require "namagem/version"

Gem::Specification.new do |s|
  s.name        = "namagem"
  s.version     = Namagem::VERSION
  s.platform    = Gem::Platform::RUBY
  s.authors     = ["Nugroho Herucahyono"]
  s.email       = ["xinuc@xinuc.org"]
  s.homepage    = ""
  s.summary     = %q{Gem tanpa fitur}
  s.description = %q{Gem tanpa fitur.}

  s.add_development_dependency "rspec"

  s.rubyforge_project = "namagem"

  s.files         = `git ls-files`.split("\n")
  s.test_files    = `git ls-files -- {test,spec,features}/*`.split("\n")
  s.executables   = `git ls-files -- bin/*`.split("\n").map{ |f|
File.basename(f) }
  s.require_paths = ["lib"]
end
```

Selanjutnya kita dapat melakukan build gem tersebut.

```
gem build namagem.gemspec
```

Pada umumnya, library ruby opensource dipublish oleh penulisnya ke rubygems.org, sehingga siapapun dapat menginstall dan menggunakannya dengan mudah. Kita dapat mempublish gem kita dengan perintah sebagai berikut:

```
gem push namagem-0.0.1.gem
```

Testing

Testing merupakan aspek yang sangat penting dalam programming. Testing dapat digunakan untuk memastikan apakah kode yang kita tulis bekerja sesuai dengan yang kita harapkan.

Di ruby, ada banyak sekali testing framework yang tersedia. Diantaranya: test-unit, minitest, rspec, shoulda dan lain-lain. Pada sesi ini, kita akan membahas unit testing dengan menggunakan minitest, yang merupakan test framework bawaan ruby (>= 1.9).

Unit Testing with Minitest

Untuk menggunakan minitest, kita cukup me-require library tersebut dengan `require 'minitest/autorun'` dan membuat testcase untuk mengetes kode yang kita buat dengan assertion.

```
# person.rb
class Person
  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end

  def fullname
    @first_name + " " + @last_name
  end
end

# person_test.rb
require './person'
require 'minitest/autorun'

class PersonTest < MiniTest::Unit::TestCase
  def setup
    @person = Person.new "John", "Smith"
  end

  def test_create_a_correct_fullname
    assert_equal "John Smith", @person.fullname
  end
end
```

atau kita dapat menuliskan test dengan gaya rspec:

```
# person_test.rb
require './person'
require 'minitest/autorun'

describe Person do
  before do
    @person = Person.new "John", "Smith"
  end

  it "should create a correct fullname" do
    @person.fullname.must_equal "John Smith"
  end
end
```

kemudian kita jalankan dengan `ruby person_test.rb`, yang akan menghasilkan

```
Run options: --seed 6446

# Running tests:

.

Finished tests in 0.000741s, 1349.7900 tests/s, 1349.7900 assertions/s.

1 tests, 1 assertions, 0 failures, 0 errors, 0 skips
```

yang berarti bahwa ada 1 test yang dijalankan tanpa failure.

Introduction to TDD

TDD (test driven development) adalah metode pembuatan software dengan menuliskan test terlebih dahulu, baru kemudian menulis kode implementasinya. Setelah kode implementasi memenuhi testnya, kode kemudian di refactor ke dalam bentuk yang lebih modular.

Misalnya kita akan menulis kode untuk mengubah besaran temperatur dari celsius ke fahrenheit dan sebaliknya. Pertama, kita akan membuat list fitur dari kode tersebut, yaitu:

- dapat mengubah temperatur dari celsius ke fahrenheit
- dapat mengubah temperatur dari fahrenheit ke celsius

kemudian, kita buat test case nya terlebih dahulu, sebagai berikut:

```
# temperature_converter_test.rb

require 'minitest/autorun'

describe TemperatureConverter do
  it "can convert temperature from celsius to fahrenheit" do
    TemperatureConverter.new(10).celsius_to_fahrenheit.must_equal 50
  end

  it "can convert temperature from fahrenheit to celsius" do
    TemperatureConverter.new(50).fahrenheit_to_celsius.must_equal 10
  end
end
```

Jika kita jalankan test ini, maka tentu akan gagal. Baru kemudian kita buat implementasinya sampai testnya berhasil.

```
# temperature_converter.rb

class TemperatureConverter
  def initialize(temp)
    @temp = temp
  end

  def celsius_to_fahrenheit
    @temp * 9 / 5 + 32
  end

  def fahrenheit_to_celsius
```

```
      (@temp - 32) * 5 / 9
    end
  end
```