Ruby y Rails

Temario

- Introducción
- Hello Word y Literales
- Expresiones
- Bloques
- Programación orientada a objetos (OOP)
- Introspección
- Rake
- Gemas

Ruby / Introducción

- Ruby es un lenguage de propósito general
- Es orientado a objetos
- Es de tipado dinámico
- Introspección y alteración dinámica

Ruby / Introducción

- **Se puede ejecutar en distintas plataformas**
- **Diferentes implementaciones**
  - YARV (C standard)
  - JRuby -> JVM
  - IronRuby -> .NET
- **Software libre, licencia GPL**

Ruby / Hello Word y Literales

- **Hello world**

```
puts "Hello world !"
```

- **Literales**

```
'A unicode string'

['an array with', 3, 'elements']

{ :color => :green, 'model' => 1970, 5 => 8}

/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i
```

Ruby / Expresiones

```ruby
1 + 3

"I'm an object, ".+ " right?"

Math::PI

[:monday, :tuesday] << :wednesday
```

- Los operadores son syntax sugar

Ruby / Bloques

- Los bloques en ruby son porciones de código asociados a un scope de variables
- Son objetos
- Bloque de una línea:

```
3.times { puts "Hello" }
```

Ruby / Bloques

- Pueden recibir parametros

```
[9, 14, 16, 19, 2, 4].select do |n|
  n < 10
end
```

- Ejemplo de implementación del método select

```
def select(&block)
  selected = []

  each do |n|
    v = yield n
    selected << n if v
  end

  selected
end
```

Ruby / Bloques

- **Son útiles para código transaccional**
- **Un método crea una transacción, ejecuta el código dentro del bloque (yield) y por último hace rollback**

```
# Yields block inside a transaction
def transactional
  t = Transaction.new
  yield
  t.rollback
end
```

- **Ejecutamos código transaccional dentro del bloque**

```
transactional do
  preload_data
  insert test_cases
  assert true
end
```

Ruby / OOP

- Ruby tiene soporte "casi" completo de OOP

```
class Vehicle

  attr_accessor :registration

  def initialize registration
    @registration = registration
  end

  def on; end
end
```

- El ejemplo anterior define una clase con:
  - getter / setter
  - initializer
  - instance method

Ruby / OOP

- Se crea una nueva instancia

```
my_buggie = Vehicle.new 'AHC 2990'

my_buggie.on
```

moove-iT

Ruby / OOP

- Herencia
- La clase Spaceship hereda de Vehicle

```
class Spaceship < Vehicle
  def orbit; end
end
```

- Accediendo a atributos de la clase padre

```
ufo = Spaceship.new "UFO - 0X"
ufo.registration
# => "UFO - 0X"
```

Ruby / OOP

- Ruby no soporta **herencia múltiple**
- En su lugar provee el concepto de **mixins**

```
module LandVehicle

  def park; end
end
```

```
module WaterCraft
  attr_accessor :displacement # cc

  def moor; end
end
```

- El comportamiento de módulos puede ser **combinado** con comportamiento de clase

Ruby / OOP

- **Definimos una clase que:**
  - hereda de Vehicle
  - incluye funcionalidad de los módulos

```
class Argo < Vehicle
  include WaterCraft
  include LandVehicle

  attr_accessor :wheels

  def initialize registration, wheels
    super registration
    @wheels = wheels
  end
end
```

Ruby / OOP

- **Creamos una nueva instancia de la clase(mixed)**

```
my_argo = Argo.new 'MXC-23', 6
my_argo.displacement = 890
my_argo.park
```

- **La instancia puede acceder a métodos de los módulos y a métodos desde su herencia**

Ruby / OOP

- public, protected y private son sólo métodos

```
public
def draw
end

def serialized_data
end
```

```
protected
def draw_axes
end
```

```
private
def calculate_bounds
end
```

```
protected :draw_legends
```

Introspección

- Ruby tiene soporte para observación y modificación de la estructura de un objeto

```
my_argo.class
# => Argo

my_argo.instance_of? Argo
# => true
```

```
my_argo.methods
```

Ruby / Introspección

- Los métodos de instancia son accesibles desde la clase

```
Argo.public_instance_methods

my_argo.public_methods
```

- Los métodos public, private o protected pueden ser devueltos con la llamada al método correspondiente
- Los métodos de instancia son accesibles desde la instancia en sí

```
my_argo.methods
```

Ruby / Introspección

- Verificar si un objeto puede llamar a un método

```
pie_chart.respond_to? :draw
# => true
```

- Si el método es privado (private), respond_to siempre devuelve false

```
pie_chart.respond_to? :calculate_bounds
# => false
```

- Los métodos se pueden invocar dinámicamente

```
pie_chart.send :draw
```

Ruby / Introspección

- Se puede definir métodos dinámicamente

```
Argo.send :define_method, :print_wheels do
  puts @wheels
end
```

- Las variables de instancia pueden ser asignadas desde afuera del objeto

```
my_argo.instance_variable_set :@wheels, 8
```

- Como las variables de instancia, las constantes y variables de clase también pueden ser asignadas desde afuera del objeto

Ruby / Rake

- Ruby build tool
- Funcionalmente similar a make o ant
- Soporta dependencia de tareas, documentación e incluye un conjunto de tareas predefinidas

```
desc "Clear temporary files"
task :clear do
  FileUtils.rm(Dir['tmp/*'])
end
```

- Invocar una tarea es tan sencillo como ejecutar la siguiente línea desde una consola

```
$> rake tmp:clear
```

Ruby / Gems

- Las gemas son un sistema de manejo de paquetes
- Cada paquete en ruby es una gema
- Ruby provee una utilidad para manejar gemas
- Maneja dependencias entre los paquetes
- Cada gema tiene información sobre ella misma

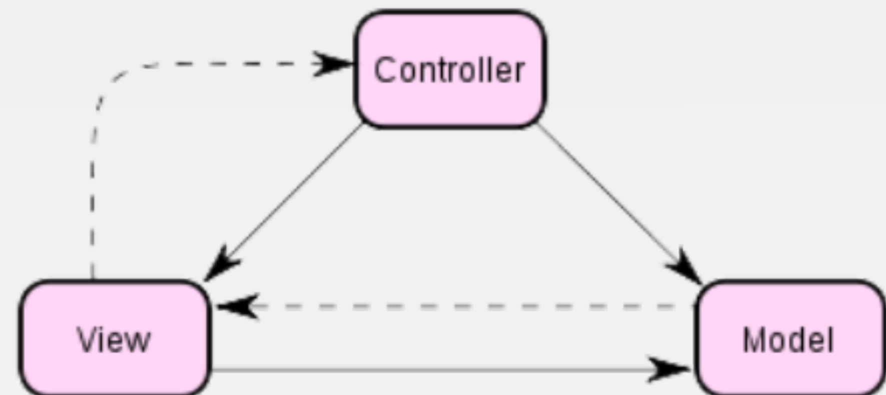
```
Gem::Specification.new do |s|
  s.name          = "capistrano"
  s.version       = Capistrano::Version.to_s
  s.platform      = Gem::Platform::RUBY
  s.authors       = ["Jamis Buck", "Lee Hambley"]
  s.summary       = "Easy deployment"
end
```

¿Qué es Ruby on Rails?

- **Rails es un framework para desarollo de aplicaciones web**
- **Se basa en el patrón de diseño MVC**
- **Principios que sigue**
  - Convención en lugar de configuración (Convention over configuration o CoC)
  - DRY (Don't Repeat yourself)
- **Soporte para varios idiomas (I18n)**
- **Muy buena documentación**

MVC (Modelo Vista Controlador)

- El modelo es el responsable de interactuar con la base de datos
- El controlador es el encargado de interactuar con los modelos para obtener la información que se necesita en las vistas
- Las vistas son las que le muestran la información al usuario



ORM

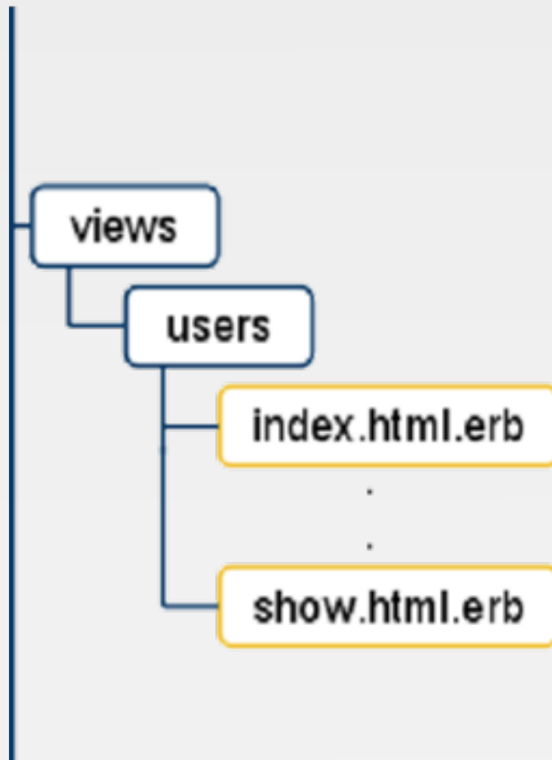
- El ORM nos permite escribir código ruby en lugar de SQL y mapear los resultados en modelos
- También nos permite cambiar fácilmente de motor de base de datos ya que las consultas están escritas en ruby y son interpretadas por el ORM
- El ORM de Rails es ActiveRecord por defecto
- Por convención los nombres de las tablas son el plural del nombre del modelo

Convention over Configuration

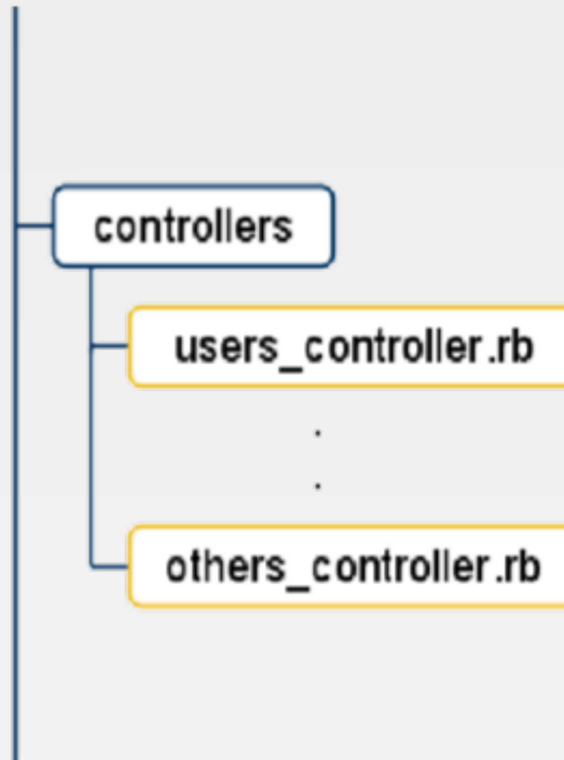
- Cuando se siguen las convenciones pre-establecidas es todo mucho más sencillo
- En lugar de configurar seguir las convenciones
- Hay que hacer algunas configuraciones, dentro de las cuales se encuentra el acceso a la base de datos
- En lugar de tener muchas líneas de configuración, es recomendable aprender la convención (la misma va a servir para proyectos que sigan el mismo patrón)

Rails Convention

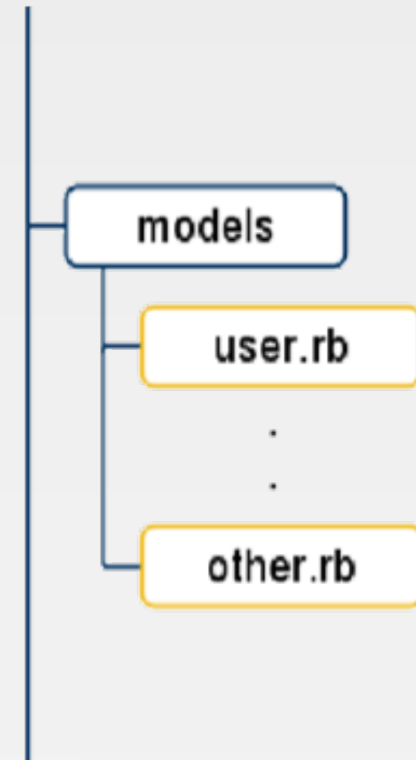
Vistas



Controladores



Modelos



DRY (Don't repeat yourself)

- **La representación de un concepto en la aplicación debe ser único y no puede ser ambiguo**

```
class User < ActiveRecord::Base; end

class UsersController < ApplicationController
  def index
    @users = User.where(:active => true)
  end
end

class  DashboardController < ApplicationController
  def show
    @dashboard = Dashboard.find params[:id]

    @users = @dashboard.users.where(:active => true)
                             .order("users.activity DESC")
  end
end
```

moove-iT

DRY (Don't repeat yourself)

- **La representación de un concepto en la aplicación debe ser único y no puede ser ambiguo**

```
class User < ActiveRecord::Base
  scope :active, where(:active => true)
  scope :most_active, active.order("users.activity DESC")
end

class UsersController < ApplicationController
  def index
    @users = User.active
  end
end

class  DashboardController < ApplicationController
  def show
    @dashboard = Dashboard.find params[:id]
    @users = @dashboard.users.most_active
  end
end
```

Vistas

- **Las vistas son las que le van a mostrar la información al usuario**
- **Para mantener las vistas sin codigo repetido (DRY) rails nos ofrece los partials**
- **Cuando hay código complejo en una vista, el mismo debería ser pasado a un helper**
- **Rails nos provee muchos helpers pre-definidos, algunos de ellos son**
  - time_ago_in_words
  - pluralize
  - Y muchos otros...

Resumen

- **Cómo crear un nuevo proyecto con Rails**
- **La estructura de un proyecto Rails**
- **MVC (Modelo Vista Controlador)**
- **ORM (ActiveRecord)**
- **Principios en los que se basa Rails**
  - DRY
  - Convención en lugar de configuración

GRACIAS!

¿Preguntas?

Ruby y Rails