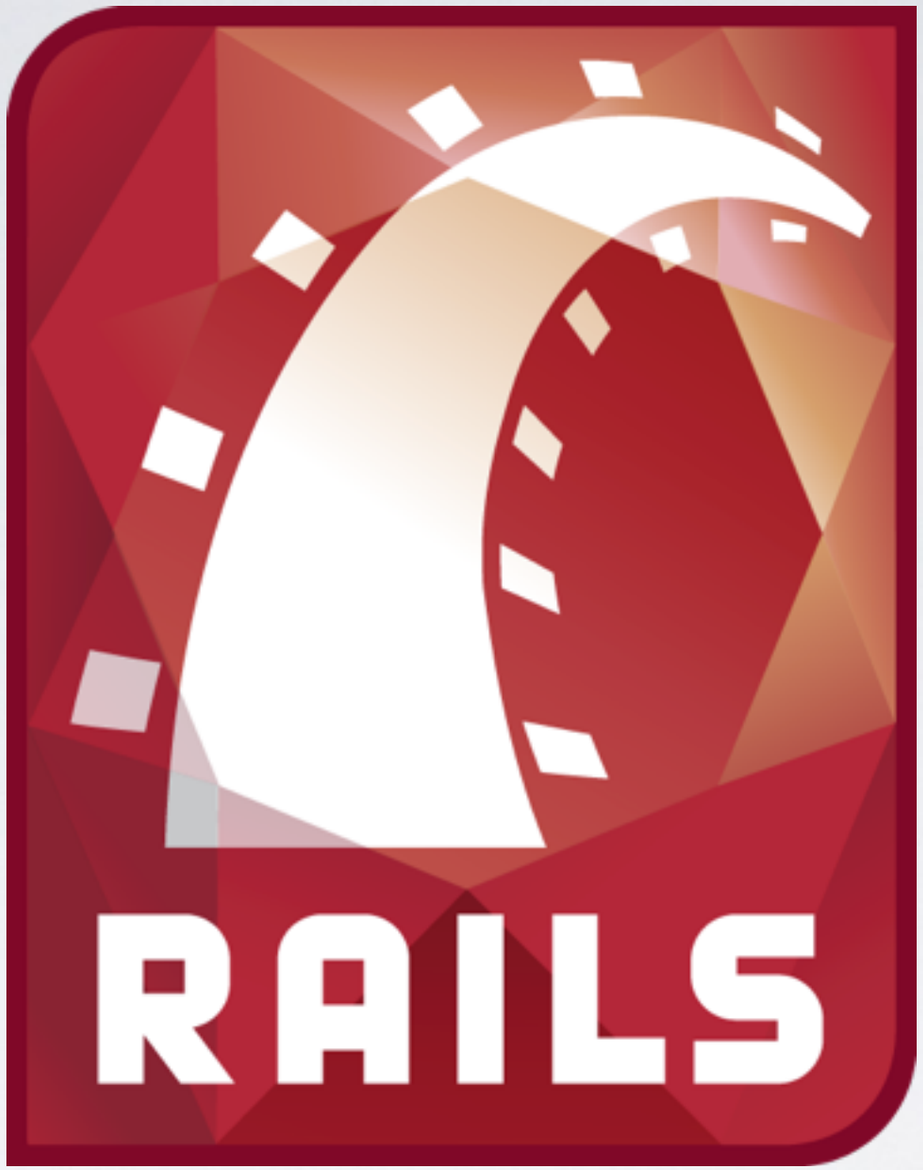


RUBY ON RAILS

"Web development that doesn't hurts"

PARTE II: ROR

- Contenido
    - Introducción al framework
    - Componentes clave
    - Testing
    - JavaScript // CoffeeScript
    - LiveController
    - RailsAvanzado
    - APIs y Servicos

INTRODUCCIÓN A RAILS

- Creado por David Hanson (HDD)
- 2004 se publica la primera beta
- La empresa de HDD, 37Signals, es la encargada de gestionar el core
- Sale a partir del proyecto Basecamp de 37S
- La combinación con Ruby hace un framework muy potente y flexible. Reduce el time 2 market.

MITOS SOBRE RAILS

- Rails es un framework demasiado nuevo
- Rails es difícil de desplegar en sistemas
- Rails no tiene comunidad
- No escala
- No es multi-thread (comparémoslo con el siguiente…)
- No es concurrente

USAN RAILS

EA MicroSites

YellowPages

Twitter
algo le queda

NewYork Times

Groupon

GitHub

Ask.fm

Basecamp

Hulu

ThemeForrest

SlideShare

Scribid

VK

Shopify

change.org

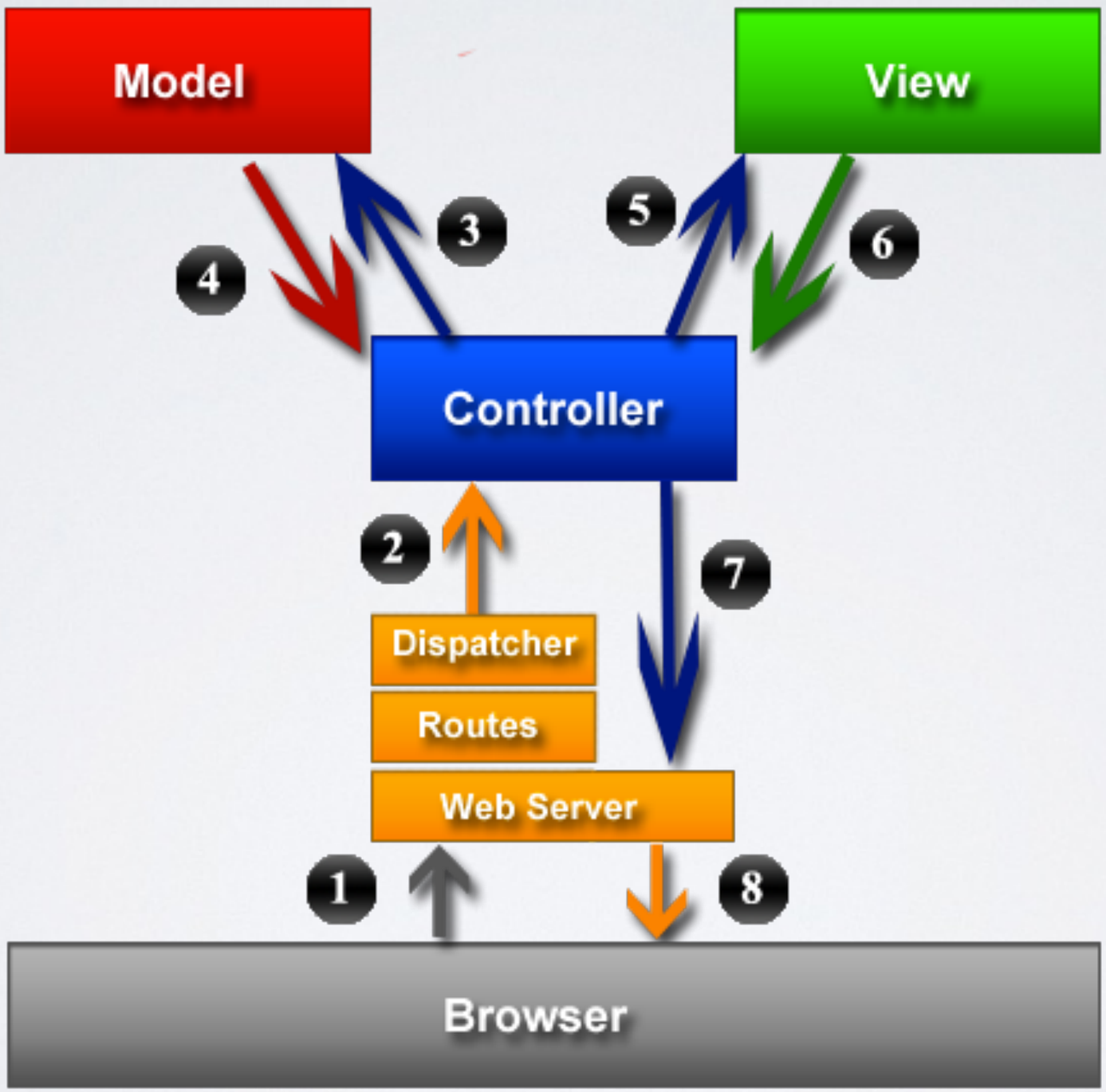
CARATERÍSTICAS

- Arquitectura MCV
- ORM (ActiveRecord)
- Convención sobre configuración
- Principios DRY
- Embedded Ruby para las vistas (*.erb)
- jQuery como framework de JavaScript por defecto

CARATERÍSTICAS

- Testing completo incluido
- Conexiones permanentes con LiveController
- Gestión de la caché básica o avanzada (Russian doll caching)
- Turbolinks
- Sistema de comandos muy completo

MVC

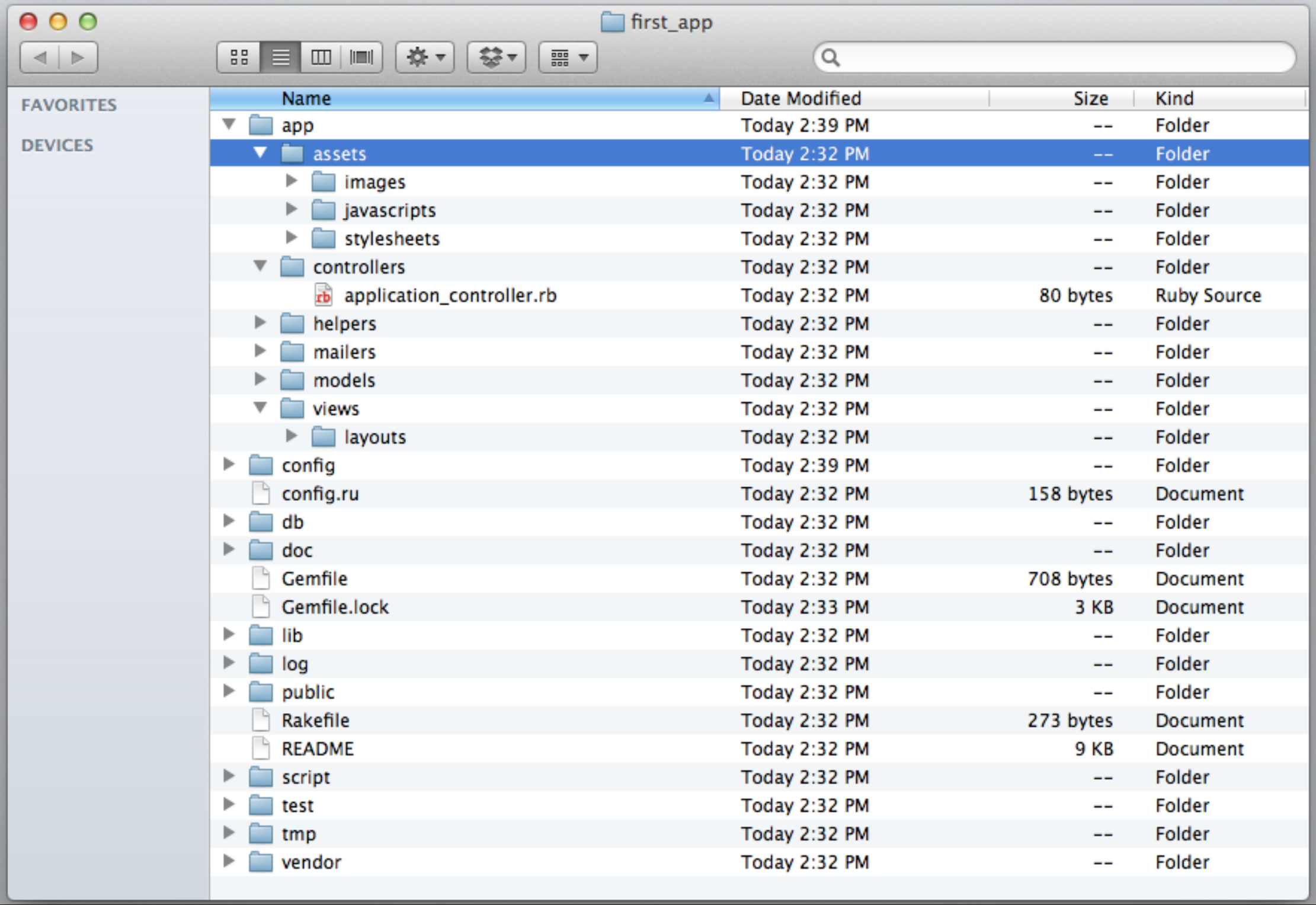

Model
View
3
4
5
6
Controller
2
7
Dispatcher
Routes
Web Server
1
8
Browser

CREACIÓN DE UNA APP

```
# Creación de una app
$ rails new nombre_app
……
```

```
# Variaciones
-m “plantilla.rb”     # usa un template
-d “base_de_datos”    # modifica el adapter
–edge                 # usa vers. edge
```

FICHEROS


first_app
FAVORITES
DEVICES
Name
Date Modified
Size
Kind
app Today 2:39 PM -- Folder
assets Today 2:32 PM -- Folder
images Today 2:32 PM -- Folder
javascripts Today 2:32 PM -- Folder
stylesheets Today 2:32 PM -- Folder
controllers Today 2:32 PM -- Folder
application_controller.rb Today 2:32 PM 80 bytes Ruby Source
helpers Today 2:32 PM -- Folder
mailers Today 2:32 PM -- Folder
models Today 2:32 PM -- Folder
views Today 2:32 PM -- Folder
layouts Today 2:32 PM -- Folder
config Today 2:39 PM -- Folder
config.ru Today 2:32 PM 158 bytes Document
db Today 2:32 PM -- Folder
doc Today 2:32 PM -- Folder
Gemfile Today 2:32 PM 708 bytes Document
Gemfile.lock Today 2:33 PM 3 KB Document
lib Today 2:32 PM -- Folder
log Today 2:32 PM -- Folder
public Today 2:32 PM -- Folder
Rakefile Today 2:32 PM 273 bytes Document
README Today 2:32 PM 9 KB Document
script Today 2:32 PM -- Folder
test Today 2:32 PM -- Folder
tmp Today 2:32 PM -- Folder
vendor Today 2:32 PM -- Folder

ENTORNOS & CONF

- En la carpeta environments encontramos diferentes configuraciones. Tres por defecto.
  - Development
  - Test
  - Production
- database.yml guarda la configuración de la bbdd

SCAFFOLDING

- Técnica de andamiaje
  - generación rápida de los principales componentes de un objeto
  - se indica el objeto, sus atributos y el tipo de dato
  - fichero de “migración” , modelo, controlador, vistas, tests, helpers, javascript y css

SCAFFOLDING

```
# Creación de un scaffold
$ rails g scaffold modelo [atributos…]
```

```
# otros generadores
$ rails g layout nombre
$ rails g model nombre attributos
$ rails g controller nombre métodos
```

FICHEROS DE MIGRACIÓN

- Contienen las instrucciones para trabajar sobre las tablas de las base de datos
- Son independientes del SGBD
- Gestiona la marcha atrás en caso de error con el rollback

FICHEROS DE MIGRACIÓN

```
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :titulo
      t.text :contenido

      t.timestamps
    end
  end
end
```

FICHEROS DE MIGRACIÓN

- Las migraciones además gestionan:
  - carga de datos programadas en el seed.rb
  - la versión de la bbdd actual y controla que migración debe ejecutar
  - create, drop, truncate de la base de datos

FICHEROS DE MIGRACIÓN

```
# Podemos usar los siguientes comandos rake

$ rake db:create      # crea la bbdd
$ rake db:migrate     # carga migraciones
$ rake db:seed        # relleno de bbdd
$ rake db:rollback    # marcha atrás
```

BLOG EN 20 MINUTOS

```
# Vamos a crear nuestra primera app rails

$ rails new my_blog
$ cd my_blog
# configuramos la base de datos
$ rails g scaffold Post title:string
content:text
$ rails g scaffold Comment content:text
post:references
$ rake db:create
$ rake db:migrate
$ rails server
```

COMPONENTES DEL FRAMEWORK

PRINCIPALES

- ActiveRecord
  - Validaciones, asociaciones, callback, interfaz de consulta
- ActionView
  - Layouts, renders, parciales y helpers
- ActionController
- ActiveSupport

ACTIVE RECORD (AR)

- Representa la M del MVC. Nos proporciona una capa de acceso a los datos, lógica y persistencia.
- Baso en un patrón diseñado por Martin Fowler
- Sigue los principios de convención sobre configuración.

AR CONVENCIÓN

- Nombre de tablas: plural con guión bajo para separar las palabras.

- Modelos: Singular y con las palabras juntas y capitalizadas.

- Claves primarias: campo **id**

- Claves foráneas: nombre de la tabla en singular + "_id"

AR CONVENCIÓN

- created_at / updated_at para la gestión de "versiones"

- lock_version: provoca que se use la estrategia de bloqueo optimista en un modelo

- type: usado para hacer STI (single table inheritance)

- asociacion_**type**: almacena relaciones polimórficas.

AR CONVENCIÓN

- Los modelos que extiendan de ActiveRecord::Base se entiende que siguen la convención.

- Podemos modificar la convención:
  - self.table_name indica el nombre de la tabla
  - self.primary_key indica el nombre de la clave primaria

AR BÁSICO

- Operaciones CRUD
  - Create
  - Raed
  - Update
  - Delete

AR BÁSICO

```
# CRUD: Create

Post.create(title: “…”, content: “…”)

user = User.new
user.name = “Pablo”
user.email = “pablo@pabloformoso.com”
user.save

user = User.new do |u|
   …
end
```

AR BÁSICO

```
# CRUD: Create

Post.create(title: “…”, content: “…”)

user = User.new
user.name = “Pablo”
user.email = “pablo@pabloformoso.com”
user.save

user = User.new do |u|
   …
end
```

AR BÁSICO

```
# CRUD: Read

Post.all
post_uno = Post.first
ultimo_post = Post.last

post = Post.find_by(title: “…”)
post = Post.find_by(id: 2)

Post.where(id: 1).order(“created_at DESC”)
```

AR BÁSICO

```
# CRUD: Update

post = Post.find_by(id: 2)

post.update(title: “nuevo”)
post.update_attribute(:title, “nuevo”)
post.update_attributes(title: “nuevo”,
content: “…”)

Post.update_all “title = ‘general’”
```

AR BÁSICO

```
# CRUD: DELETE

post = Post.find_by(id: 2)
post.destroy
```

AR QUERIES

- La interfaz para consultas nos permite lanzar sentencias sencillas y complejas sin mucho esfuerzo

- Haciendo uso del eager loading estas consultas están optimizadas en su gran mayoría.

- No implica que un join o includes puedan generar una query lenta.

AR QUERIES

- Entre los principales métodos nos encontramos:
  - where, uniq, distinct
  - order, reverse_order
  - group, from, offset, limit
  - joins, includes

AR QUERIES

```
# Recuperación de un solo registro

Post.take

Post.first
Post.last

Post.find 1
Post.find_by title: “…”
```

AR QUERIES

```
# Recuperación de varios registros

Post.find([1,10])       # 2 valores
Post.find([1..10])

Post.take(2)

Post.first(2)
Post.last(3)
```

AR QUERIES

```
# Recuperación de registros por batch

Post.find_each do |p|
  enviar_por_email(p)
end

Post.find_each(batch_size: 2000) do |p|
  enviar_por_email(p)
end
```

AR QUERIES

```
# Recuperación condicionada

Post.where(published: true)
Post.where(“published == 1”)

Post.joins(:comments).where(
   comments: {published: true})

Post.where(created_at:
   (Time.now.midnight - 1.day)..Time.now.midnight)

Post.where(comments_count: [2,6])

Post.where.now(published: true)
```

AR QUERIES

```
# Recuperación de campos
Post.select(“title”)
Post.select(:title).distinct

# Limites y offsets
Post.limit(10)
Post.limit(2).offset(10)

# Agrupaciones
Post.where(published: true)
   .group(“comments_count”)
```

AR QUERIES

```
# Joins

Post.joins(:category, :comments)
```

```
SELECT posts.* FROM posts
  INNER JOIN categories ON posts.category_id = categories.id
  INNER JOIN comments ON comments.post_id = posts.id
```

AR ASSOCIATIONS

- Nos ayudan a establecer diferentes asociaciones entre modelos.

- Simplifica operaciones de dependencia a la hora de crear, leer o destruir un recurso.

- Hace de interfaz a la hora de recuperar una colección de datos.

AR ASSOCIATIONS

```
# Supongamos estos dos modelos
class Customer < AR::Base
end

class Order < AR::Base
end

# Normalmente para eliminar un usuario
# borraríamos todos los pedidos y luego
# el usuario
```

AR ASSOCIATIONS

```
# Supongamos estos dos modelos
class Customer < AR::Base
  has_many :orders, dependent: :destroy
end

class Order < AR::Base
  belongs_to :customer
end

# A través de las dependencias al borrar el
# comprador se borraran todos los pedidos
# :)
```

AR ASSOCIATIONS

```
# Tipos de asociaciones

belongs_to
has_one
has_many
has_one :through
has_many :through
has_and_belongs_to_many
```

AR ASSOCIATIONS

```
class Customer < AR::Base
   has_many :orders, dependent: :destroy
end

class Order < AR::Base
   belongs_to :customer
end
```

El **belongs_to** nos indica el modelo en el que se espera la clave foránea en la relación, para este caso siguiendo la convención **customer_id**

AR ASSOCIATIONS

El through nos permite establecer una relación a través de un modelo. En el ejemplo médico, cita, paciente, se ilustra la vinculación.

```
class Doctor < AR::Base
   has_many :citas
   has_many :pacientes, through: :citas
end

class Cita < AR::Base … end
class Paciente < AR::Base … end
```

AR ASSOCIATIONS

```
class Doctor < AR::Base
   has_many :citas
   has_many :pacientes, through: :citas
end

class Cita < AR::Base
   belongs_to :doctor
   belongs_to :paciente
end

class Paciente < AR::Base
   has_many :citas
   has_many :doctores, through: :citas
end
```

AR ASSOCIATIONS

- Las relaciones polimórficas esconden una potencia, flexibilidad y tolerancia a cambios. Entraremos en detalle con las nociones avanzadas sobre Rails.