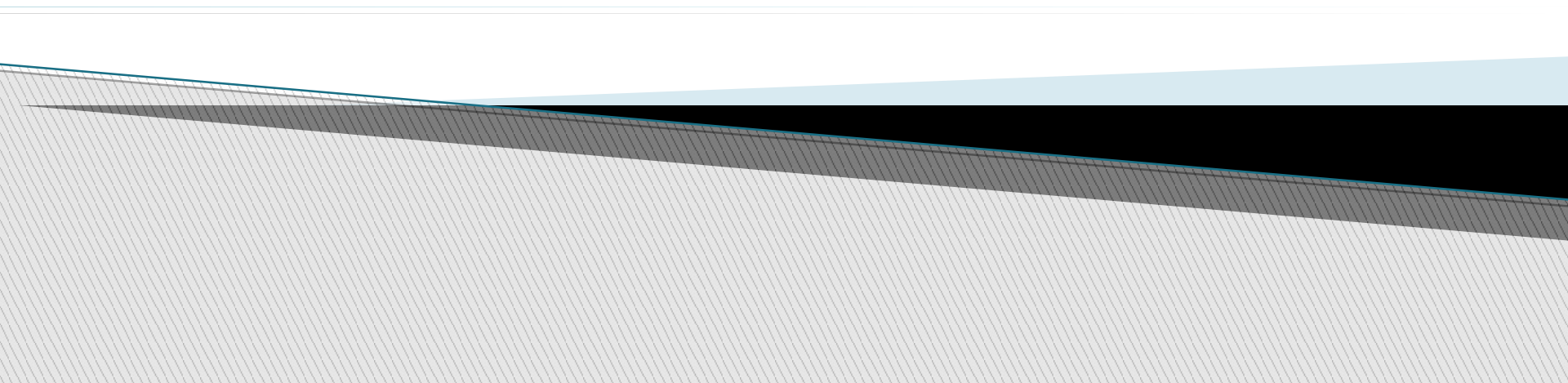


# Linguagem de Programação para Web

Ruby On Rails – parte 1  
Prof. Tales Bitelo Viegas



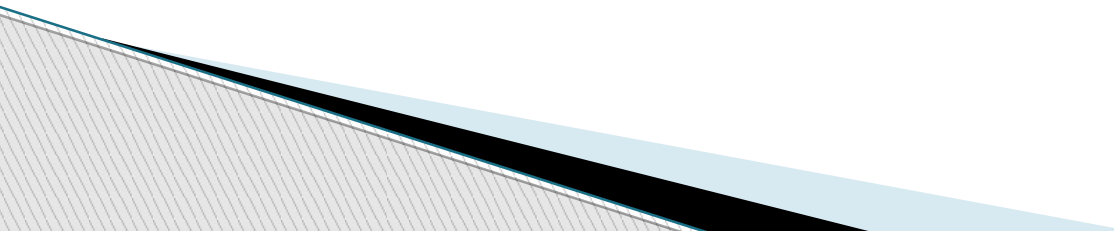
# Ruby vs Rails

- ▶ Ruby é uma linguagem de programação
  - Como C, Java, Python, etc
  - Pode ser utilizado para programar qualquer coisa
- ▶ Rails é um Framework
  - Promove funcionalidades web comuns
  - Foco em seu aplicativo, não em detalhes de “baixo nível”

# Rails é um Framework Web

- ▶ Desenvolva, instale e mantenha aplicações web dinâmicas
- ▶ Escrito utilizando Ruby
- ▶ Extremamente flexível, com rápido desenvolvimento

# Tecnologias

- ▶ HTML – Exibição
  - ▶ JavaScript – interação
  - ▶ CSS – Formatação
  - ▶ Rails – Cria a aplicação web
- 

# Arquitetura

- ▶ Terminologia
  - DRY – *Don't Repeat Yourself*
- ▶ Arquitetura do Rails
  - MVC (*Model View Controller*)
  - ORM (*Object Relational Mapping*)
  - RESTful (*Representational State Transfer*)

# DRY

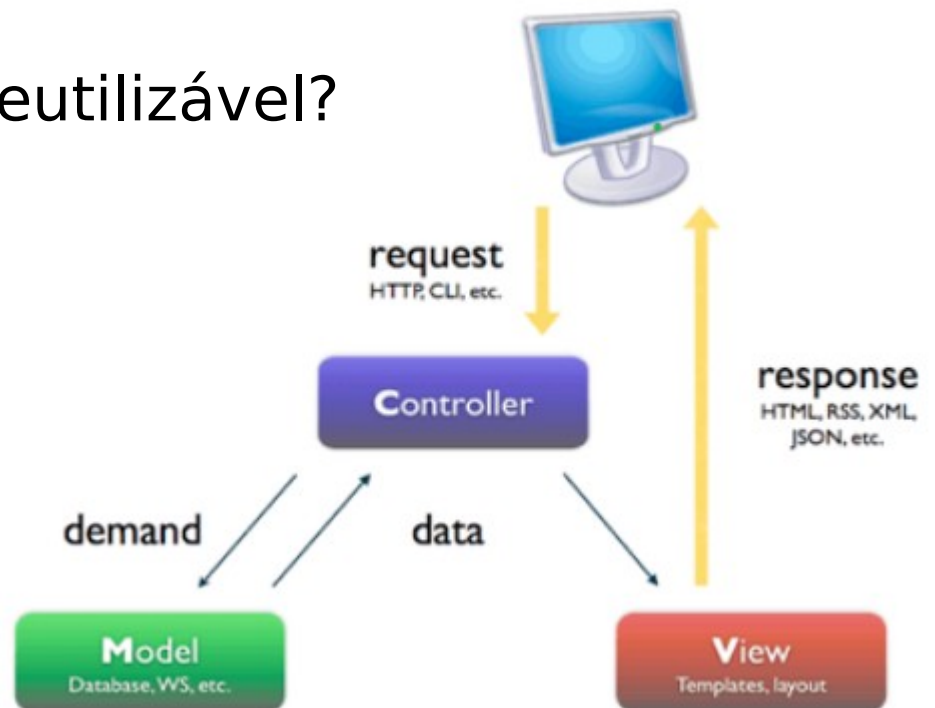
- ▶ **Don't Repeat Yourself**



- ▶ Reutilize, não reinvente

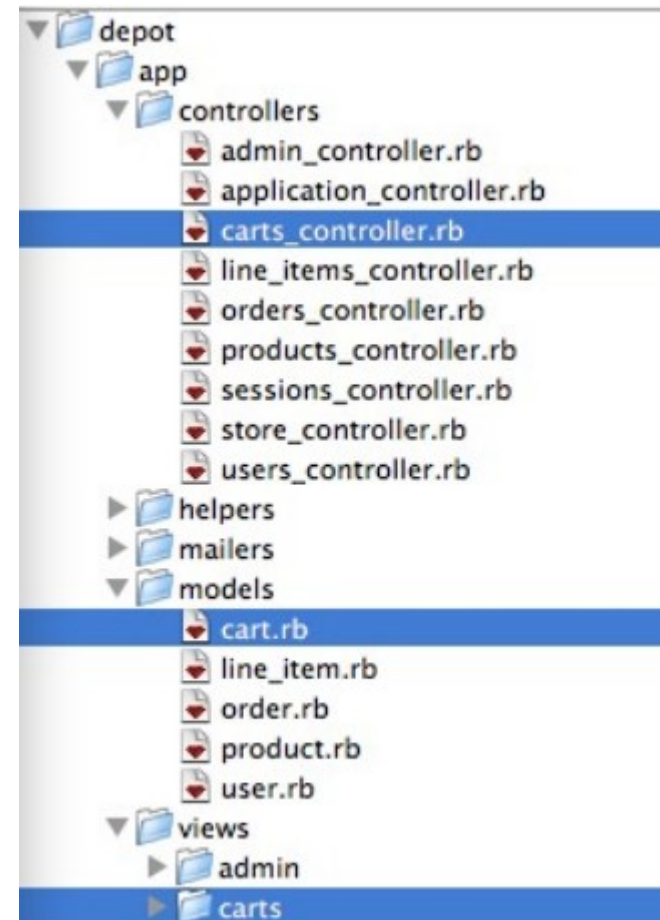
# Model-View-Controller

- ▶ Isola a “Lógica do Domínio”
  - Posso ver?
    - ▢ *View*
  - É lógica de negócio?
    - ▢ *Controller*
  - É uma classe lógica reutilizável?
    - ▢ *Model*



# Model-View-Controller

- ▶ Gerado pelo Rails
  - Agrupado por pastas
  - Conectadas “AutoMagicamente”
    - ▢ Models
    - ▢ Views
    - ▢ Controllers
  - Múltiplas Views por Controller





# Modelos de Banco de Dados

- ▶ Armazena e acessa grande quantidade de dados
- ▶ Tabela
  - Colunas (nome, tipo, modificador)
  - Linhas

Column	Type	Modifiers
id	integer	not null default nextval('users_id_seq'::regclass)
uuid	character varying(255)	
fb_id	bigint	
created_at	timestamp without time zone	
updated_at	timestamp without time zone	
email	character varying(255)	
password_hash	character varying(255)	
hometown	character varying(255)	
username	character varying(255)	

# SQL

- ▶ Structured Query Languages
  - Como falar com a base de dados

```
SELECT *  
  FROM Book  
 WHERE price > 100.00  
 ORDER BY title;
```

# SQL - Operações

- ▶ Inserção
- ▶ Consulta
- ▶ Atualização e remoção
- ▶ Criação e modificação da estrutura da base de dados

# Mapeamento Objeto-Relacional

- ▶ Mapeia a base de dados para objetos Ruby
- ▶ ActiveRecord

```
>> userVariable = User.where(:name => "Bob")
```

Generates:

```
SELECT      \"users\".* FROM \"users\"  
WHERE      (name = 'bob')
```

```
>> userVariable.name
```

```
=> Bob
```

# Mapeamento Objeto-Relacional

```
>> userVariable = User.where(:name => "Bob")
```




models/user.rb

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

- ▶ A classe **User** herda de ActiveRecord::Base

# Mapeamento Objeto-Relacional

```
>> userVariable = User. where(:name => "Bob")
```



- ▶ **where** é o **método** que procura na base de dados AutoMagicamente na tabela User (se você a criou)

# RESTful

- ▶ Representational State Transfer
- ▶ O estado da mensagem importa
  - Estado diferente = mensagem diferente



*"You Again?"*



*"You Again?"*

# RESTful

- ▶ Servidores levam em conta a maneira como você os acessa
- ▶ Métodos HTTP
  - GET
  - PUT
  - POST
  - DELETE



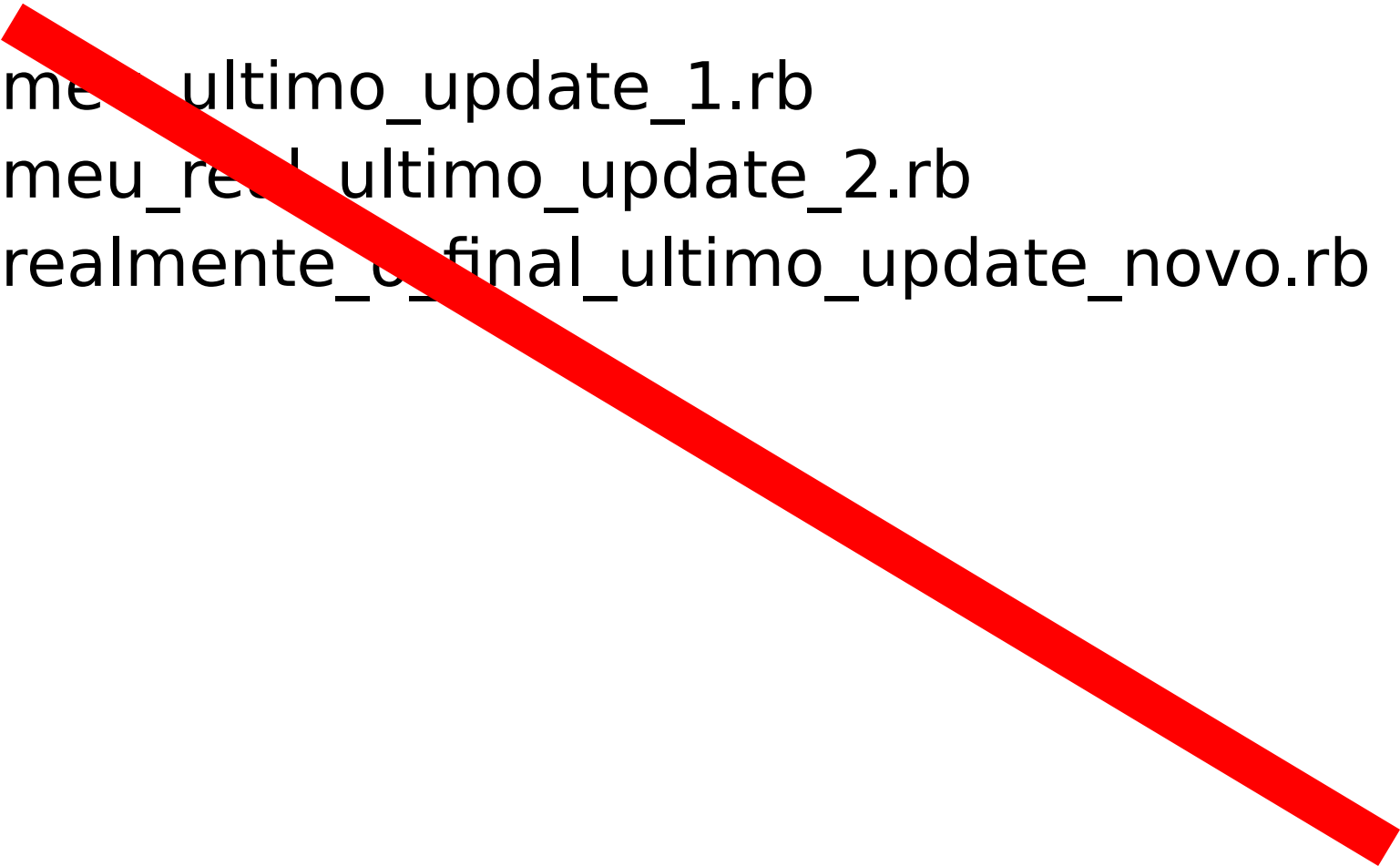
# RESTful

- ▶ Rails mapeia ações aos métodos HTTP
  - GET - index, show, new
  - PUT - update
  - POST - create
  - DELETE - destroy

# Ambiente de Trabalho

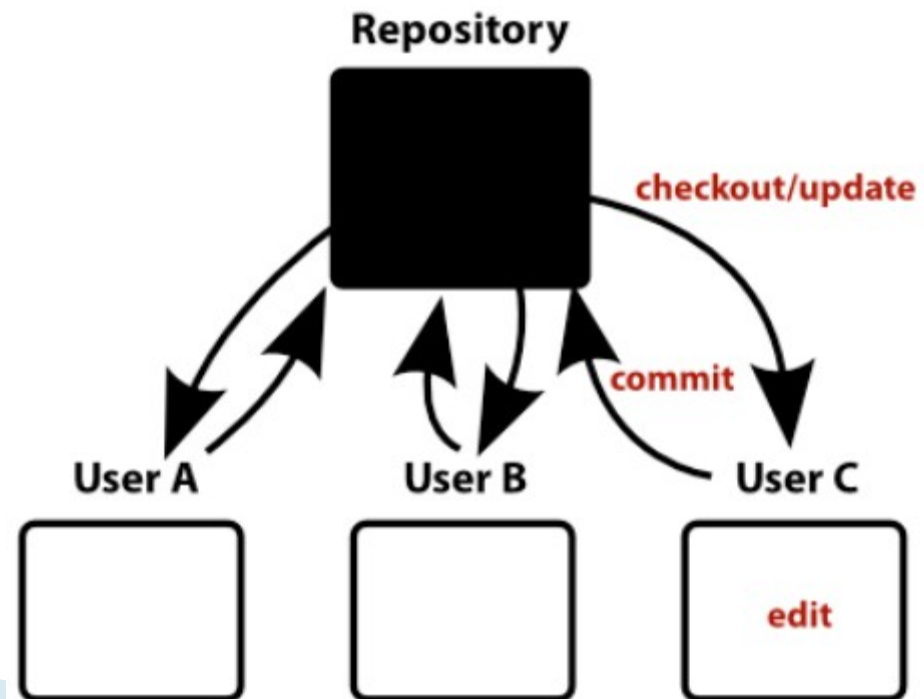
- ▶ Controle de versão – manter seu código seguro
- ▶ RubyGems – utilização de código de terceiros no seu aplicativo
- ▶ Bundler – gerenciamento de dependências

# Controle de Versão

- ▶ meu\_ultimo\_update\_1.rb
  - ▶ meu\_real\_ultimo\_update\_2.rb
  - ▶ realmente\_o\_final\_ultimo\_update\_novo.rb
- 

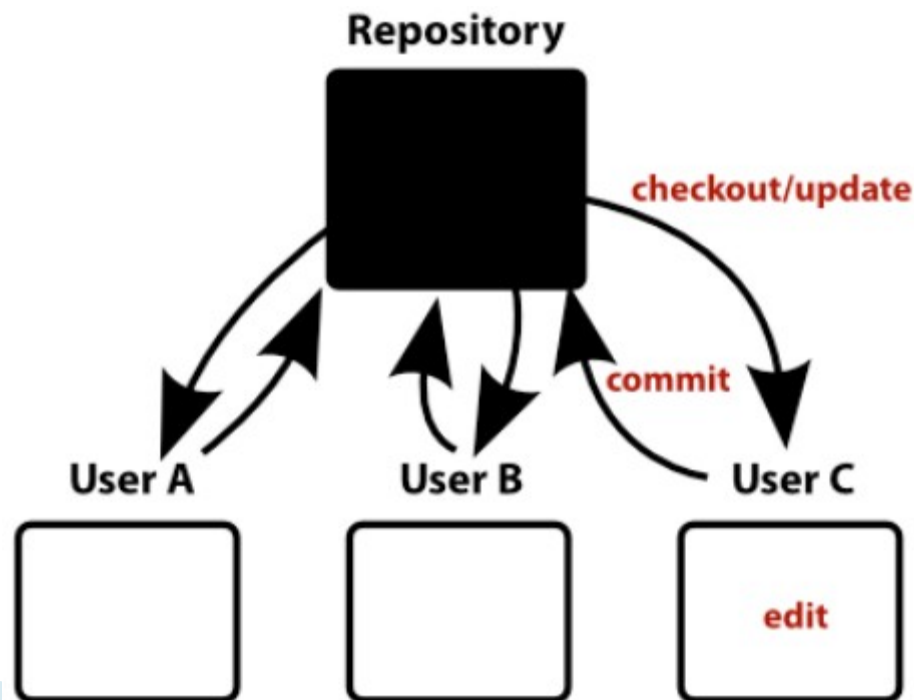
# Controle de Versão

- ▶ Documente o que foi alterado
- ▶ Veja as mudanças no código durante o tempo
- ▶ Volte a um estado anterior
- ▶ Trabalhe em um time



# Controle de Versão

- ▶ Git (recomendado)
- ▶ SVN
- ▶ Mercurial
- ▶ Perforce
- ▶ Entre outros...



# Github

lib/keytar/key\_builder.rb

5

lib/keytar/key\_builder.rb

View file @ 7266237

```
... @@ -31,6 +31,7 @@ module KeyBuilder
31 31  module Ext
32 32    # creates class level getter and setter methods for the defaults for config
33 33    DEFAULTS.keys.each do |key|
34 +   # TODO: re-write without eval
35 35    eval %{
36 36      def #{key}({key}_input = :key_default)
37 37        @@#{key} = DEFAULTS[:#{key}] unless defined? @@#{key}
... @@ -80,8 +81,8 @@ module KeyBuilder
80 81    options.keys.each do |key|
81 82      options["key_#{key}".to_sym] = options[key] if key.to_s !~ /^key_/
82 83    end
83 84    - options.keys.each do |key|
84 85      - eval("@@#{key} = options[key]")
85 86      + options.each do |key, value|
86 87      +   self.send( key , value) if self.respond_to? key
87 88    end
88 89    end
89 90    alias :keyfig :key_config
```

► <http://github.com>

# RubyGems

- ▶ Gems
  - Pacotes contendo códigos externos
- ▶ Rubygems gerencia estes pacotes

# Bundler

- ▶ Gerenciamento de dependências
- ▶ Instalação

```
gem install bundler
```

- ▶ Gemfiles
  - Especifica as dependências

```
source :rubygems

gem 'rails', '3.0.4'
gem 'unicorn', '3.5.0'
gem 'keytar'
```



# Bundler

- ▶ Instalação

```
bundle install
```

- ▶ Instala todas as gems listadas no arquivo gemfile
- ▶ Útil para instalações entre sistemas diferentes

# IDE

- ▶ Mac: Textmate
- ▶ Windows: Notepad++
- ▶ Sublime2 (editor de texto),
- ▶ Eclipse & Aptana RadRails

# Ruby e Rails

- ▶ Hashes são comumente utilizados como parâmetros aos métodos
  - Options é um parâmetro opcional

```
text_area(object_name, method, options = {})
```

Returns a textarea opening and closing tag set tailored for accessing a specified attribute (identified by method) on an object). Additional options on the input tag can be passed as a hash with options.

## Examples

```
text_area(:post, :body, :cols => 20, :rows => 40)
# => <textarea cols="20" rows="40" id="post_body" name="post[body]">
#    #{@post.body}
# </textarea>

text_area(:comment, :text, :size => "20x30")
# => <textarea cols="20" rows="30" id="comment_text" name="comment[text]">
#    #{@comment.text}
# </textarea>

text_area(:application, :notes, :cols => 40, :rows => 15, :class => 'app_input')
# => <textarea cols="40" rows="15" id="application_notes" name="application[notes]" class="app_input">
#    #{@application.notes}
# </textarea>
```

# Instalando o Rails

- ▶ Windows

- <http://railsinstaller.org/>

- ▶ Linux

- `gem install rails`

# Criando a primeira aplicação

- ▶ rails new blog
- ▶ Isto irá criar um novo aplicativo Rails chamado “blog”, utilizando a base de dados SQLite (padão, usar -d para outras bases)
- ▶ Para instalar as dependências:
  - bundle install

# Estrutura de Pastas

Pasta	Propósito
app/	Contém os controles, modelos, views e demais arquivos da aplicação
config/	Arquivos de configurações
db/	Contém o modelo corrente da base de dados
doc/	Documentação da aplicação
lib/	Bibliotecas ou módulos adicionais da aplicação
log/	Arquivos de log (registro) da aplicação
public/	A única pasta vista de fora como está. Contém os arquivos estáticos
script/	Arquivos script do Rails que inicia a aplicação
test/	Unit tests, fixtures e outros testes
tmp/	Arquivos temporários
vendor/	Todos os arquivos de terceiros

# Configurando a base de dados

- ▶ Arquivo config/database.yml
- ▶ Um database por ambiente (development, test, production)
- ▶ Acertar a senha
- ▶ Criar o banco de dados com:
  - rake db:create

# Hello Rails

- ▶ Após a base de dados criada, podemos começar a ver nossa aplicação
- ▶ O Rails possui um servidor de aplicação próprio
- ▶ Para inicializá-lo:
  - rails server



# Hello Rails



## Welcome aboard

You're riding Ruby on Rails!

[About your application's environment](#)

### Getting started

Here's how to get rolling:

1. Use `rails generate` to create your models and controllers

To see all available options, run it without parameters.

2. Set up a default route and remove *public/index.html*

Routes are set up in *config/routes.rb*.

3. Create your database

Run `rake db:create` to create your database. If you're not using SQLite (the default), edit *config/database.yml* with your username and password.

### Browse the documentation

[Rails Guides](#)

[Rails API](#)

[Ruby core](#)

[Ruby standard library](#)

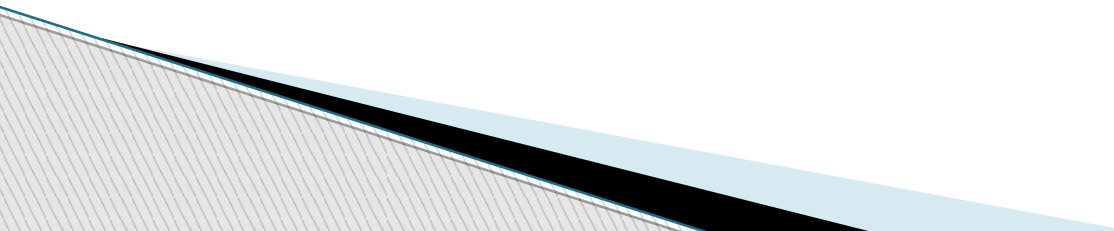
# Alterando a Home Page

- ▶ Podemos gerar um novo Controller
- ▶ Comando:
  - rails generate controller home index
- ▶ Cria um controller chamado “home”, com um método chamado index
- ▶ Editar app/views/home/index.html.erb

# Alterando a Home Page

- ▶ Alterar a Rota padrão
- ▶ Editar o arquivo config/routes.rb
- ▶ Alterar root :to para:
  - root “home#index”

# Criando rapidamente...

- ▶ Scaffold
  - ▶ Cria rapidamente Model, View e Controller para um novo recurso, em uma única operação.
  - ▶ rails generate scaffold Post name:string title:string content:text
- 

# Executando uma Migration

- ▶ CreatePosts

```
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :name
      t.string :title
      t.text :content

      t.timestamps
    end
  end
end
```

- ▶ rake db:migrate

# Adicionando um Link

- ▶ Editar app/views/home/index.html.erb
- ▶ Alterar para:

```
<h1>Hello, Rails!</h1>  
<%= link_to "My Blog", posts_path %>
```

# Model Post

- ▶ A classe Model para o Post é uma classe relativamente Simples:

```
class Post < ActiveRecord::Base  
end
```

# Model Post - Validações

- ▶ Podemos adicionar algumas validações a este modelo

```
class Post < ActiveRecord::Base
  validates :name, :presence => true
  validates :title, :presence => true,
              :length => { :minimum => 5 }
end
```

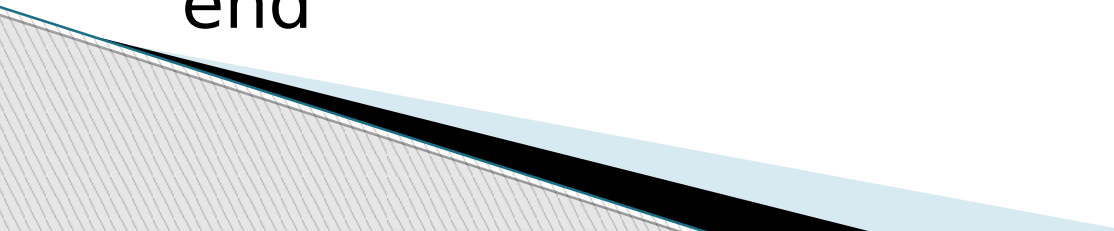


# Controller Posts

- ▶ Listando todos os Posts

```
def index
  @posts = Post.all

  respond_to do |format|
    format.html # index.html.erb
    format.json { render :json => @posts }
  end
end
```



# View Post

- ▶ app/views/posts/index.html.erb

```
<h1>Listing posts</h1>

<table>
  <tr>
    <th>Name</th>
    <th>Title</th>
    <th>Content</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

  <% @posts.each do |post| %>
    <tr>
      <td><%= post.name %></td>
      <td><%= post.title %></td>
      <td><%= post.content %></td>
      <td><%= link_to 'Show', post %></td>
      <td><%= link_to 'Edit', edit_post_path(post) %></td>
      <td><%= link_to 'Destroy', post, :confirm => 'Are you sure?', :method => :delete %></td>
    </tr>
  <% end %>
</table>

<br />

<%= link_to 'New Post', new_post_path %>
```

# Customizando o Layout

- ▶ app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog</title>
  <%= stylesheet_link_tag "application" %>
  <%= javascript_include_tag "application" %>
  <%= csrf_meta_tags %>
</head>
<body style="background: #EEEEEE;">

  <%= yield %>

</body>
</html>
```

# Controller Posts

- ▶ Criando novos posts. Duas ações:
  - Instanciar um objeto vazio e passar para a view “new.html.erb”

```
def new
  @post = Post.new

  respond_to do |format|
    format.html # new.html.erb
    format.json { render :json => @post }
  end
end
```

```
<h1>New post</h1>
```

```
<%= render 'form' %>
```

```
<%= link_to 'Back', posts_path %>
```

# Controller Posts

## ▶ Partials

- Porções de HTML e código Rails que podem ser reaproveitados.
- Exemplo: `_form.html.erb`

# Controller Posts

- ▶ Criando novos posts
  - Chamada HTTP POST que chama o método create

```
def create
  @post = Post.new(params[:post])

  respond_to do |format|
    if @post.save
      format.html { redirect_to(@post,
                               :notice => 'Post was successfully created.' ) }
      format.json { render :json => @post,
                           :status => :created, :location => @post }
    else
      format.html { render :action => "new" }
      format.json { render :json => @post.errors,
                           :status => :unprocessable_entity }
    end
  end
end
```