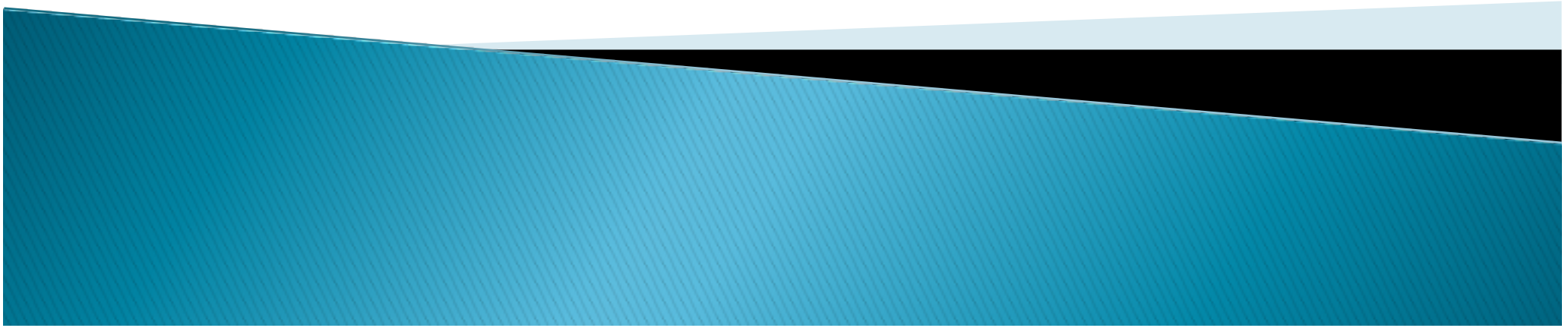
Linguagem de Programação para Web

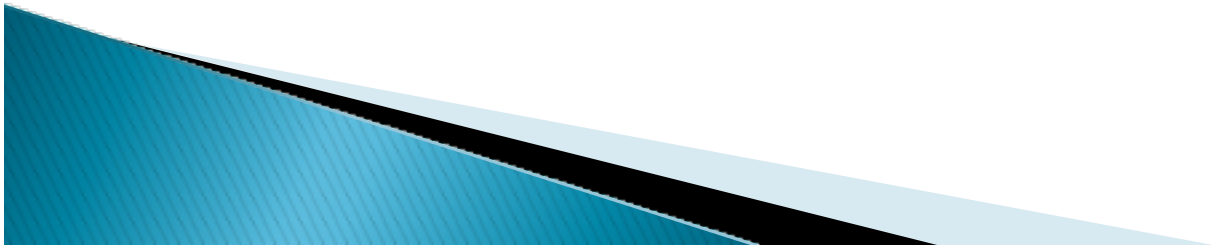
Ruby On Rails – parte 3 – Controllers

Prof. Tales Bitelo Viegas



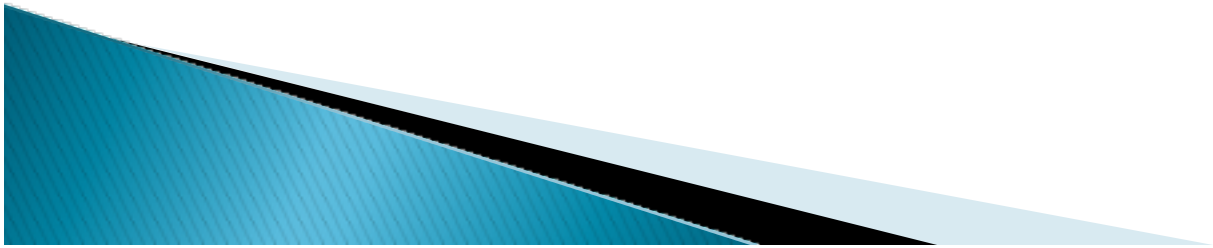
Controllers

- Responsável por receber e responder a uma requisição
- O controller irá receber a requisição, buscar ou salver dados de um modelo e usar uma view para criar uma saída.
- Se o seu controller necessita fazer coisas um pouco diferentes não é um problema.



Métodos e Ações

- Um controller é uma classe que herda de ApplicationController e possui métodos como qualquer outra classe
- Quando sua aplicação recebe uma requisição, o mecanismo de rotas determina qual controller e ação executar.
- A partir deste momento, o Rails cria uma instância do controller e executa o método da ação especificada



Métodos e Ações

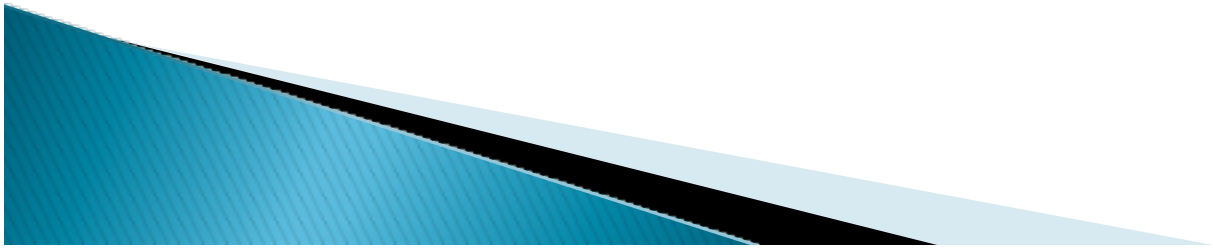
```
class ClientsController < ApplicationController
  def new
  end
end
```

- Se um usuário for para /clients/new em sua aplicação para adicionar um novo cliente, Rails irá criar uma instância de ClientsController e executar o método “new”.
- Por padrão, o Rails irá renderizar a view localizada em /app/views/clients/new.html.erb

Parâmetros

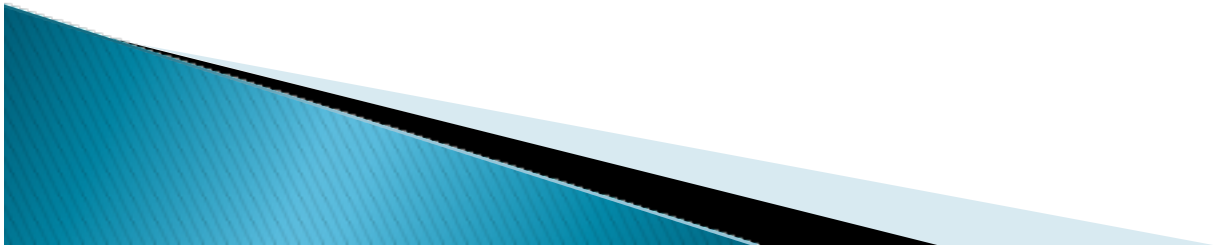
- Os dados enviados pelo usuário (seja via GET ou POST) são enviados em um hash chamado params

```
def index
  if params[:status] == “activated”
    @clients = Client.activated
  else
    @clients = Client.unactivated
  end
end
```



Parâmetros

```
def create
  @client = Client.new(params[:client])
  if @client.save
    redirect_to @client
  else
    render :action => “new”
  end
end
```



Sessões

- Configurado em:
  - config/initializers/session_store.rb

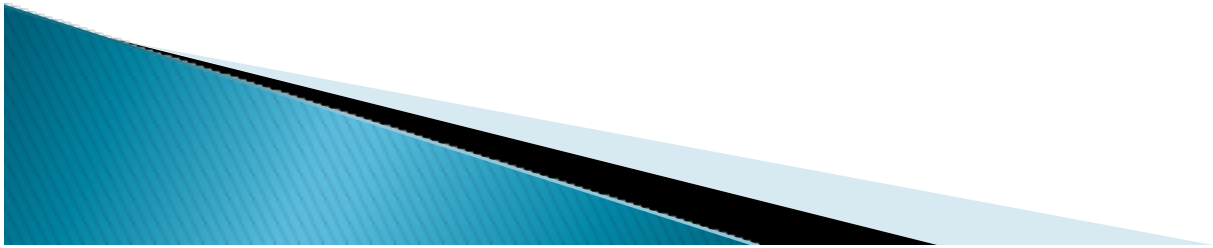
- Utilizada diretamente:

```
class ApplicationController < ActionController::Base

  private

  # Finds the User with the ID stored in the session with the key
  # :current_user_id This is a common way to handle user login in
  # a Rails application; logging in sets the session value and
  # logging out removes it.
  def current_user
    @_current_user ||= session[:current_user_id] &&
      User.find_by_id(session[:current_user_id])
  end
end
```

Sessions

- Armazenando valores na sessão

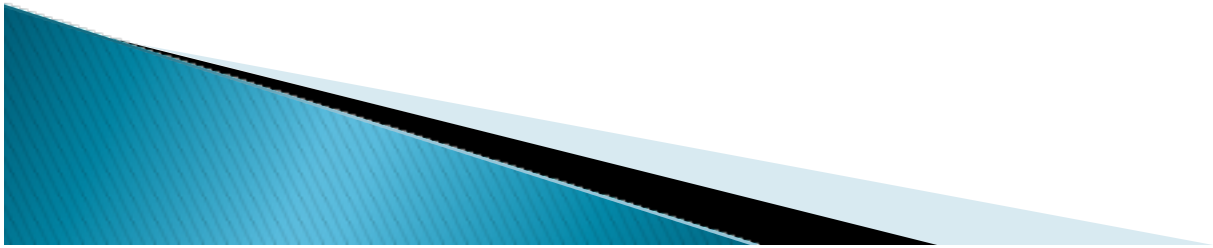
```
class LoginsController < ApplicationController
  # "Create" a login, aka "log the user in"
  def create
    if user = User.authenticate(params[:username], params[:password])
      # Save the user ID in the session so it can be used in
      # subsequent requests
      session[:current_user_id] = user.id
      redirect_to root_url
    end
  end
end
```



Sessions

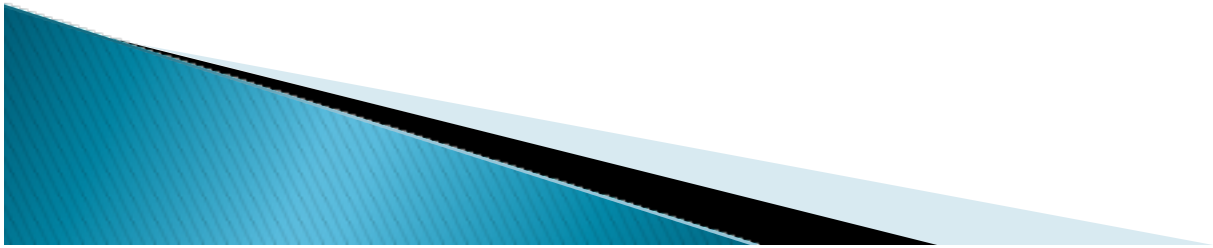
- Removendo valores da sessão (atribuir nil)

```
class LoginsController < ApplicationController
  # "Delete" a login, aka "log the user out"
  def destroy
    # Remove the user id from the session
    @_current_user = session[:current_user_id] = nil
    redirect_to root_url
  end
end
```



Renderizando XML ou JSON

```
class UsersController < ApplicationController
  def index
    @users = User.all
    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @users}
      format.json { render :json => @users}
    end
  end
end
```



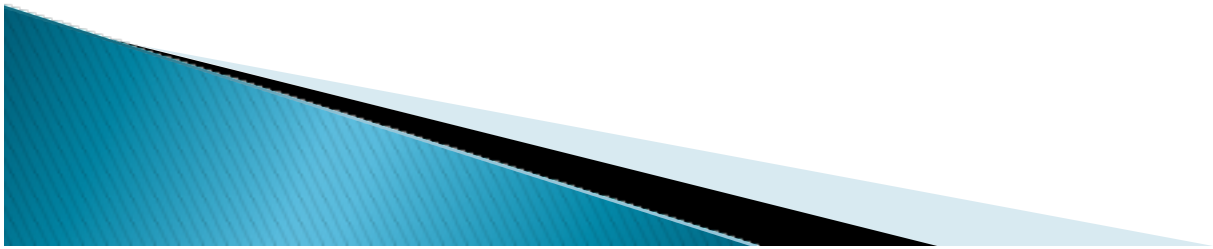
Filtros

- Filtros são métodos a serem executados antes, depois ou junto com um controller

```
class ApplicationController < ActionController::Base
  before_filter :require_login

  private

  def require_login
    unless logged_in?
      flash[:error] = "You must be logged in to access this section"
      redirect_to new_login_url # halts request cycle
    end
  end

  # The logged_in? method simply returns true if the user is logged
  # in and false otherwise. It does this by "booleanizing" the
  # current_user method we created previously using a double ! operator
  # Note that this is not common in Ruby and is discouraged unless you
  # really mean to convert something into true or false.
  def logged_in?
    !!current_user
  end
end
```

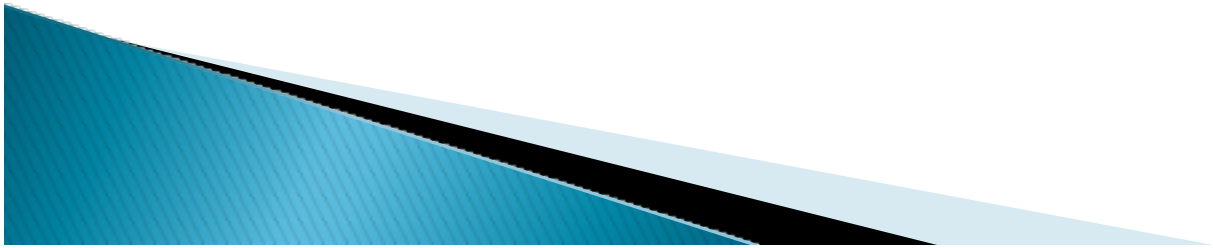
Filtros

```
class LoginsController < ApplicationController
  skip_before_filter :require_login, :only => [:new, :create]
end
```



Request e Response

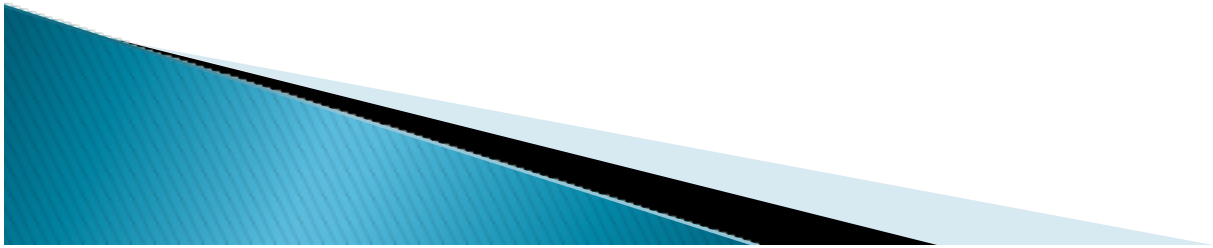
- Cada controller recebe os objetos request e response, da requisição corrente



Roteamento de Requisições

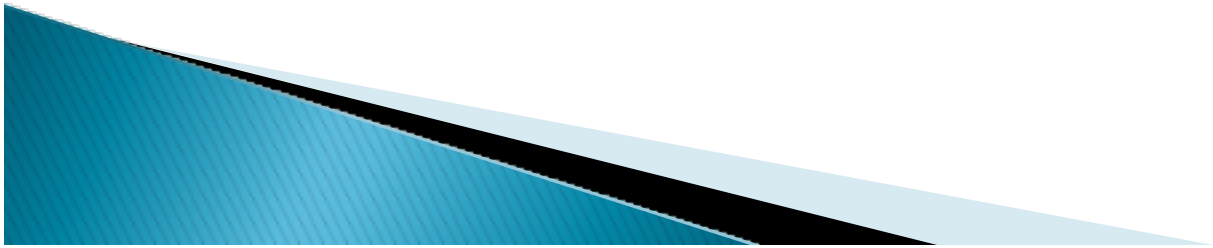
- O Rails possui um roteador de requisições que reconhece a URL informada e envia para um controller.

- Rotas são mapeadas no arquivo /config/routes.rb da sua aplicação



Gerando Paths e URLS a partir do código

```
@patient = Patient.find(17)

<% link_to “Patient Record”, patient_path(@patient) %>
```



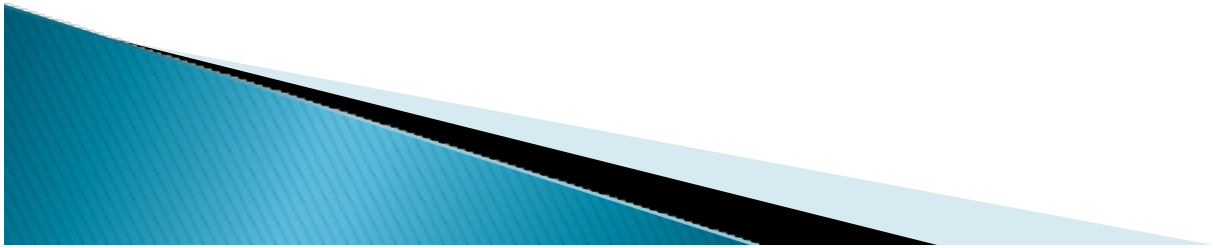
Rails default – Resources

- Criação das requisições RESTful para os controllers
  - resources :photos

HTTP	Path	Action	Usada para
GET	/photos	index	Mostrar lista de fotos
GET	/photos/new	new	Retorar HTML para criar nova foto
POST	/photos	create	Criar uma nova foto
GET	/photos/:id	show	Exibir uma foto
GET	/photos/:id/edit	edit	Retornar um HTML para editar foto
PUT	/photos/:id	update	Atualizar uma foto
DELETE	/photos/:id	destroy	Excluir uma foto

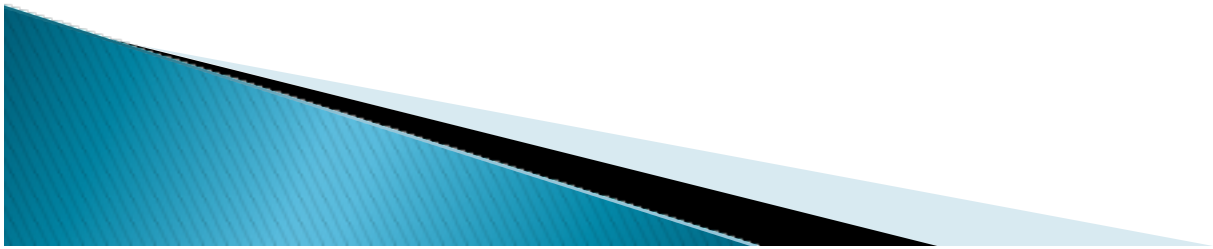
Paths e URLs

- photos_path
- new_photo_path
- edit_photo_path(:id)
- photo_path(:id)



Resource específico

- match “profile” => “users#show”



Resources Alinhados

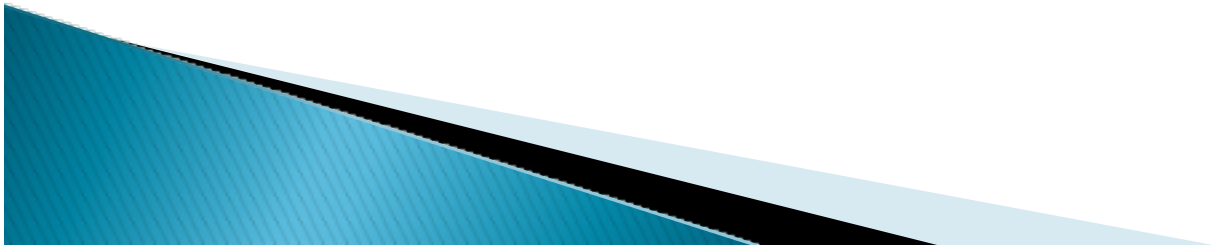
- resources :magazines do
  resources :ads
  end

HTTP	Path	Action
GET	/magazines/:magazine_id/ads	index
GET	/magazines/:magazine_id/ads/new	new
POST	/magazines/:magazine_id/ads	create
GET	/magazines/:magazine_id/ads/:id	show
GET	/magazines/:magazine_id/ads/edit	edit
PUT	/magazines/:magazine_id/ads/:id	update
DELETE	/magazines/:magazine_id/ads/:id	destroy

Adicionando mais RESTful

```
resources :photos do
   member do
      get ‘preview’
   end
end
```

/photos/1/preview



Testando as rotas

- rake routes