

Especialização em Desenvolvimento Web

Programação para Servidor 1 - Aula 1

Msc Édimo Sousa Silva
edimo.sousa@fapce.edu.br



Sumário

- Arquitetura Cliente - Servidor
- Request e Response
- Ruby
- Ruby on Rails
- Criando controllers e models
- Validações nos models
- REST
- CRUD - parte 1
- JSON Response format
- HTTP Status Code
- Refatoração
- Relacionamentos



Arquitetura Cliente-Servidor

- É a forma mais popular atualmente na web
- Adequada para lidar com vários usuários
- Cloud com AWS e Azure
- Servidor
 - Entidade que oferece o serviço
 - Web hosting, processamento, storage
- Cliente
 - Entidade que consome o serviço
 - Geralmente independente de plataforma



Servidor

- **Servidor web:** Sistema que responde a requisições HTTP.
- **Servidor de aplicação:** Sistema que responde a um ou vários tipos de requisições.
- ***Web service:*** Sistema com foco na interoperabilidade entre diferentes sistemas.



HTTP - Hypertext Transfer Protocol

- Message based model
- Baseado em requests e responses
- Stateless (requisições independentes)
- URL (Uniform Resource Locator)
- URI (Uniform Resource identifier)
 - file://home/Desktop/livro.pdf

`http://www.example.com/search?item=vw+beetle`

Protocol

Domain

Path

Parameters

HTTP - Request

- Request-line
- Zero ou vários headers seguidos por CRLF
- Uma linha em branco que indica o fim dos headers
- Opcional message-body



Requisição HTTP

- ***Request Line***

Request-Method SP Request-URI SP HTTP-Version

CRLF

- **Formada por 3 partes + Separador**

- Request-Method
- Request-URI
- HTTP-Version CRLF

- **Separador**

- SP: separator



Request Method

- Indica qual método será chamado no recurso de acordo com a URI.
 - PUT => update
 - DELETE => destroy.
- O request method é case sensitive e deve ser sempre utilizado em UPPERCASE



Request Method

- **GET:** recuperar dados do recurso especificado, não deve modificar dados.
- **HEAD:** idêntica a requisição GET, mas sem o body, é mais rápida e é utilizada para saber se o recurso está disponível.
- **POST:** enviar um recurso para o servidor.
- **PUT:** atualizar por completo um recurso.



Request Method

- **DELETE:** apagar um recurso.
- **CONNECT:** cria um túnel entre o cliente e o servidor usando SSL (Security Socket Layer).
- **OPTIONS:** informa os métodos disponíveis para o recurso em questão.
- **TRACE:** retorna o que foi enviado ao servidor, geralmente usado para debug.
- **PATCH:** editar parcialmente um recurso.



Request-URI

- **Uniform Resource Identifier**
 - Identificador único do recurso que a requisição tem como alvo.
- **Exemplos mais comuns**
 - absoluteURI: http://server/api/v1/resource
 - "*": O alvo é o servidor e não um recurso específico.
 - Host: http://server/
 - abs_path
 - api/v1/resource
 - Host: http://server/



HTTP-Version

- **HTTP-Version**

- ex: HTTP/1.1

- **CRLF**

- Carriage Return (ASCII 13, \r) Line Feed (ASCII 10, \n)

- **Exemplos de Request-Line**

- OPTIONS * HTTP/1.1
- GET /hello.htm HTTP/1.1
- POST http://server/api/v1/resource HTTP/1.1



HTTP - Request

- **Request-line**
- Zero ou vários headers seguidos por CRLF
- Uma linha em branco que indica o fim dos headers
- Opcional message-body



HTTP - Headers

- Permitem que o cliente envie informações adicionais para o servidor.
- Agrupados por contexto:
 - **Request header:** informações sobre o recurso buscado ou sobre o cliente.
 - **Response header:** informações sobre a resposta ou localização (nome, versão).
 - **Entity header:** informações sobre o body da entidade (tamanho, tipo de dado).
 - **General header:** informação válida para a request e para o response.



HTTP - Request Headers - Categorias

- Authentication
- Caching
- Client hints
- Conditionals
- Connection management
- Content negotiation
- Controls
- Cookies
- CORS
- [Saiba Mais :\)](#)



HTTP - Request Headers - Exemplos

- User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
- Host: www.server.com
- Content-Type: application/x-www-form-urlencoded
- Content-Length: length
- Accept-Language: en-us
- Accept-Encoding: gzip, deflate



HTTP - Request

- Request-line
- Zero ou vários headers seguidos por CRLF
- Uma linha em branco que indica o fim dos headers
- Opcional message-body



HTTP - Opcional message-body

Json Object

```
{  
  "name": "Rails",  
  "version": 5,  
  "front-end": {  
    "name": "Vue",  
    "css": null  
  }  
}
```

Json Array

```
[  
  {  
    "name": "Rails",  
    "version": 5  
  },  
  {  
    "name": "Spring",  
    "version": 8  
  }  
]
```

HTTP - Response

- **Status-line**
- Zero ou vários headers seguidos por CRLF
- Uma linha em branco que indica o fim dos headers
- Opcional message-body



HTTP - Response - Status-line

- HTTP-Version SP Status-Code SP Reason-Phrase CRLF
 - HTTP-Version = HTTP/1.1
 - Status-Code = 1xx, 2xx, 3xx, 4xx, 5xx
 - Reason-Phrase = Not found, Internal error, etc



HTTP - Response

- **Status-line**
- **Zero ou vários headers seguidos por CRLF**
- Uma linha em branco que indica o fim dos headers
- Opcional message-body



HTTP - Response - headers

- Accept-Ranges, Age, ETag, Location, Proxy-Authenticate, Retry-After, Server, Vary, WWW-Authenticate
- [Saiba mais :\)](#)



Ruby

- Criada em 1993
- Open-source
- Puramente orientada a objetos
- Paradigmas
 - Funcional
 - Procedural
 - Genérico



Ruby on Rails

- Criado em 2005
- Princípios
 - Convention over configuration,
 - DRY (Don't Repeat Yourself)
 - Fat models, skinny controller
- Curva de aprendizagem alta
- gems



RoR – Quem usa?



Configuração

- Back-end
 - Ruby (2.3.x)
 - Rails (5.x)
- Database
 - postgresql
- Client http
 - Postman ou Insomnia
- IDE
 - Visual Studio code ou similar



Exercício – Criando o projeto

1. Criar a pasta “Desenvolvimento/rails”
2. No terminal navegar até a pasta “Desenvolvimento/rails”
3. `$ rails new railsloja --api --database=postgresql`
4. `$ cd railsloja`
5. `$ bundle install`
6. `$ rake db:create`
7. `$ rails s`
8. no browser: localhost:3000



Exercício – Criando o controller

1. rails g controller api/v1/Products
2. Criar um método no controller

railsloja/app/controllers/api/v1/products_controller:

```
class Api::V1::ProductsController < ApplicationController
  def hello_world
    render json: {message: 'Hello API'}
  end
end
```

3. Criar uma rota no arquivo railsloja/config/routes.rb

```
Rails.application.routes.draw do
  namespace :api do
    namespace :v1 do
      get "hello", to: "products#hello_world"
    end
  end
end
```



Exercício – Listando produtos

1. Criar modelo do Produto
 - \$ rails g model Product name:string quantity:integer
 - \$ rake db:migrate
2. Criar método na classe ProductsController

- ```
def list
 render json: Product.all
end
```

3. Criar a rota correspondente em /config/routes.rb

- ```
get "hello", to: "products#hello_world"  
get "products", to: "products#list"
```



Exercício – populando o DB

1. Instalando a gem faker

- edite o arquivo “railsloja/Gemfile”

```
group :development do
  [...]
  gem 'faker'
  [...]
end
```

- \$ bundle install

2. Usando a gem

- edite o arquivo “railsloja/db/seeds.rb”

```
10.times do
  Product.create({
    name: Faker::Book.title,
    quantity: Faker::Number.number(3)
  })
end
```

- \$ rails db:seed



Exercício – Criando Produtos

1. Criar um método no controller

railsloja/app/controllers/api/v1/products_controller:

```
def create
  product = Product.new
  product.name = params[:name]
  product.quantity = params[:quantity]

  if product.save
    render json: product
  else
    render json: {"error": "product couldn't be saved"}
  end
end
```



Exercício – Criando Produtos (part 2)

1. Criar a rota correspondente em /config/routes.rb

```
Rails.application.routes.draw do
  namespace :api do
    namespace :v1 do
      get "hello", to: "products#hello_world"
      get "products", to: "products#list"
      post "products", to: "products#create"
    end
  end
end
```

2. Liberando a api para acesso externo
 - “railsloja/app/controllers/application_Controller.rb”

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :null_session
end
```



Exercício – Validações

1. Na classe Product => railsloja/app/models/product.rb

- ```
class Product < ApplicationRecord
 validates :name, presence: true
 validates :quantity, presence: true
end
```

2. Exibindo os erros no ProductController

- ```
def create
  product = Product.new
  product.name = params[:name]
  product.quantity = params[:quantity]

  if product.save
    render json: product
  else
    render json: {"errors": product.errors}
  end
end
```



REST

1. Representational State Transfer
2. Estilo Arquitetônico
 - Arquitetura + restrições
3. 6 Restrições
 - Cliente-Servidor
 - Stateless
 - Cache
 - Uniform Interface
 - Layered System
 - Code On Demand



REST - Cliente-Servidor (1)

1. Baseada no princípio de separação de preocupações
 - Quando for resolver um problema no client-side, se preocupe apenas com o client-side, e vice-versa.
2. Os componentes (cliente e servidor) devem ser capazes de evoluir de forma independente.
3. Server-side deve ser uma API (application programming interface).
 - Acessada por n tipos de clientes.



REST - Stateless (2)

1. A comunicação entre o client-side e o server-side deve ser stateless
 - Todas as requisições contém todas as informações necessárias para que elas sejam realizadas com sucesso.
2. Possibilita que o sistema cresça com maior facilidade.
 - O server-side não precisa armazenar o estado do client-side entre as requisições
 - Isso possibilita que ele atenda a mais cliente simultaneamente.



REST - Cache (3)

1. A API precisa ter cache.

- *The most efficient network request is one that does not use the network.*



REST - Uniform Interface (4)

1. Identificação dos recursos (resources)
 - Toda informação que pode ser nomeada, pode ser um recurso (imagem, arquivo, abstrações de entidades).
2. Manipulação dos recursos através de diferentes formatos
 - XML, JSON, HTML, JPEG.
3. Recursos auto-descritivos
 - Os headers devem conter informações que auxiliam na descrição dos recursos.
4. Hypermedia
 - Os clientes devem ser capazes de acessar recursos a partir de links.



REST - Layered System (5)

1. Entidades intermediadoras podem ser colocadas entre o cliente e o servidor, utilizando interface uniforme.
2. Elas podem interceptar requisições e retornar um valor salvo em cache.
3. O cliente não precisa saber de onde a resposta está vindo.



REST - Code On Demand (6)

1. Poder habilitar o cliente a receber códigos para execução.



Exercício – Apagar

1. Na classe ProductsController

- ```
def destroy
 product = Product.find_by_id(params[:id])
 product.destroy
end
```

## 2. Criar uma rota no arquivo railsloja/config/routes.rb

- ```
namespace :v1 do
  get "hello", to: "products#hello_world"
  get "products", to: "products#list"
  post "products", to: "products#create"
  delete "products/:id", to: "products#destroy"
end
```



Exercício – Buscar

1. Na classe ProductsController

- ```
def find
 render json: Product.find_by_id(params[:id])
end
```

## 2. Criar uma rota no arquivo railsloja/config/routes.rb

- ```
namespace :v1 do
  get "hello", to: "products#hello_world"
  get "products", to: "products#list"
  get "products/:id", to: "products#find"
  post "products", to: "products#create"
  delete "products/:id", to: "products#destroy"
end
```


Exercício – Atualizar

1. Na classe ProductsController

- ```
def update
 product = Product.find_by_id(params[:id])
 product.name = params[:name]
 product.quantity = params[:quantity]

 product.save

 render json: product
end
```

## 2. Criar uma rota no arquivo railsloja/config/routes.rb

- ```
get "hello", to: "products#hello_world"
get "products", to: "products#list"
get "products/:id", to: "products#find"

post "products", to: "products#create"
put "products/:id", to: "products#update"

delete "products/:id", to: "products#destroy"
```



Json Response - specifications

- **Json:api**
 - <https://jsonapi.org/>
- **JSend**
 - <https://github.com/omniti-labs/jsend>
- **Google**
 - <https://developers.google.com/search-ads/v2/reference/>
- **Rest-api-response-format**
 - github.com/cryptlex/rest-api-response-format



Json Response

- Enveloped (status-code:200)

```
{
  "data":{
    "id": 1,
    "title" : "Introduction to RoR",
    "description": "The best introduction"
  },
  "status": "SUCCESS"
}
```

- Not enveloped (status-code:200)

```
{
  "id": 1,
  "title" : "Introduction to RoR",
  "description": "The best introduction"
}
```



Success Responses - 2XX

- 200 - OK
 - A operação foi realizada com sucesso.
- 201 - Created
 - Criado com sucesso
- 204 - No Content :(
 - A operação foi realizada com sucesso e o body da response é vazio.



Error Responses - 4XX - Client Errors

- 400 - Bad Request (Data error)
 - A informação enviada está incompleta ou mal formatada.
 - ex: o atributo nome está faltando.
- 422 - Unprocessable Entity (Data error)
 - A informação enviada não atende as validações.
 - ex: o atributo senha é possui apenas 3 caracteres.



Error Responses - 4XX - Client Errors

- 404 - Not Found (Data error)
 - A informação enviada está completa e atende as validações, mas o recurso não foi encontrado.
 - ex: não existe objeto com id = 33.
- 401 - Unauthorized (Auth Error)
 - Token inválido ou não enviado
- 403 - Forbidden (Auth Error)
 - Token válido, mas sem o privilégio adequado.



Exercício 2 - response format - list

```
def list
  products = Product.all
  render json: products, status: :ok
end
```



Exercício 2 - response format - find

```
def find
  product = Product.find_by_id(params[:id])

  if product.nil?
    render json: {message: "Product not found"}, status: :not_found
  else
    render json: product, status: :ok
  end
end
```



Exercício 2 - response format - create

```
def create
  product = Product.new
  product.name = params[:name]
  product.quantity = params[:quantity]

  if product.save
    render json: product, status: :created and return
  elsif product.has_nil_fields
    error_status = :bad_request
  else
    error_status = :unprocessable_entity
  end

  render json: {message: 'Product not saved', errors: product.errors}, status: error_status
end
```

Na classe Product:

```
def has_nil_fields
  self.name.nil? || self.quantity.nil?
end
```



Exercício 2 - response format - update

```
def update
  product = Product.find_by_id(params[:id])
  product.name = params[:name]
  product.quantity = params[:quantity]

  if product.save
    render json: product, status: :ok and return
  elsif product.has_nil_fields
    error_status = :bad_request
  elsif
    error_status = :unprocessable_entity
  end

  render json: {message: 'Product not updated', errors: product.errors}, status: error_status
end
```



Exercício 2 - response format - delete

```
def destroy
  product = Product.find_by_id(params[:id])
  if product.nil?
    render json: {message: "Product not found"}, status: :not_found
  else
    product.destroy
  end
end
```



Exercício 3 - refatorar

```
class Api::V1::ProductsController < ApplicationController
  before_action :find_product, only: [:find, :update, :destroy]

  def find
    if @product.nil?
      render json: {message: "Product not found"}, status: :not_found
    else
      render json: @product, status: :ok
    end
  end
end
```

```
private
def find_product
  @product = Product.find_by_id(params[:id])
end
end
```



Exercício 3 - Strong parameters

1) Strong parameters

- Proíbe que parâmetros do controller sejam usados no model.
- O desenvolvedor tem que permitir explicitamente que quer utilizar aqueles parâmetros.

2) Exemplo

```
def product_params  
  params.permit(:name, :quantity)  
end
```

```
product = Product.new (product_params)
```

```
if @product.update(product_params)
```



Exercício 3 - Resources

```
# get "products", to:"products#index"  
# get "products/:id", to:"products#show"  
# post "products", to:"products#create"  
# put "products/:id", to:"products#update"  
# delete "products/:id", to:"products#destroy"  
resources :products
```



Relacionamentos - one-to-one

1) Criar o model Preço

- \$ rails g model Price value:float currency:string
- \$ rake db:migrate

- ```
class Product < ApplicationRecord
 has_one :price
end
```

- \$ rails generate migration AddPriceToProduct

```
class AddPriceToProduct < ActiveRecord::Migration[5.2]
 def change
 add_reference :prices, :product, index: true
 end
end
```

- \$ rails c



# Relacionamentos - one-to-Many

## 1) Criar o model Piece

- \$ rails g model Piece number:integer name:string
- \$ rake db:migrate

```
class Product < ApplicationRecord
 has_one :price
 has_many :pieces
end
```

- \$ rails generate migration AddPiecesToProduct

```
class AddPiecesToProduct < ActiveRecord::Migration[5.2]
 def change
 add_reference :pieces, :product, index: true
 end
end
```

- \$ rails c



```
class Api::V1::PiecesController < ApplicationController

 def index
 pieces = Piece.find_by product_id: params[:product_id]

 render json: pieces, status: :ok
 end

 def create
 piece = Piece.new (piece_params)
 if piece.save
 render json: piece, status: :created and return
 elsif piece.nil_fields?
 error_status = :bad_request
 else
 error_status = :unprocessable_entity
 end
 render json: {message: 'Piece not saved', errors: piece.errors}, status: error_status
 end

 private
 def find_piece
 @piece = Piece.find_by_id(params[:id])
 end





 def piece_params
 params.permit(:name, :number, :product_id)
 end
end
```

# Rotas encadeadas

```
Rails.application.routes.draw do
 namespace :api do
 namespace :v1 do
 resources :products do
 resources :pieces
 end
 end
 end
end
```



# REST - Best practices

- Use apenas resource nas URIs
  - /products 
  - /listProduct 
- Use plurais
  - /products 
  - /product 
- O método http define a ação

| Resource   | GET<br>(Read)           | POST<br>(Create)         | PUT<br>(Update)         | DELETE<br>(Delete)      |
|------------|-------------------------|--------------------------|-------------------------|-------------------------|
| /users     | Returns a list of users | Creates a new user       | Bulk update of users    | Delete all users        |
| /users/123 | Returns a specific User | Method not allowed (405) | Updates a specific user | Deletes a specific user |



# REST - Best practices part 2

- Não faça mal uso dos safe methods
  - GET, HEAD, OPTIONS e TRACE devem apenas retornar recursos, eles não devem alterar recursos
- Representar a hierarquia dos recursos através de URI
  - `/products/123/pieces/1` OK
- Versionamento da API
  - Header => Accept: application/railsloja.v2+json
  - url => GET /v2/products





# REST - Best practices part 3

- Responses significativas
  - POST, PUT e PATCH, devem retornar o recurso que foi manipulado.
  - Quando um recurso tiver sido criado, retornar 201
- Filtros, buscas e ordenações
  - Evite criar novas urls, use query parameters
  - `/products?status=ready&sort=quantity`
- HATEOAS
  - Hypermedia As Transfer Engine Of Application State



# REST - Best practices part 4

- HATEOAS
  - Hypermedia As Transfer Engine Of Application State

```
{

 "name": "John Doe",

 "self": "http://localhost:8080/users/123",

 "posts": "http://localhost:8080/users/123",

 "address": "http://localhost:8080/users/123/address"

}
```



# REST - Best practices part 5

- Stateless Autenticação e Autorização
  - Toda a requisição deve conter tudo o que precisa para ser executada de forma adequada.
  - JWT e OAuth2
- USO APROPRIADO DO STATUS\_CODE



# Referências

- [1] - Hypertext Transfer Protocol -- HTTP/1.1,  
<https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html>, 2019
- [2] - HTTP headers,  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>, 2019
- [3] - Ruby guides, <https://guides.rubyonrails.org/>, 2019
- [4] - Microsoft REST API Guidelines,  
[https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.m](https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md)  
[d](https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md), 2019
- [5] - RESTful Web Services, "O'Reilly Media, Inc.", 17 de dez de 2008.

