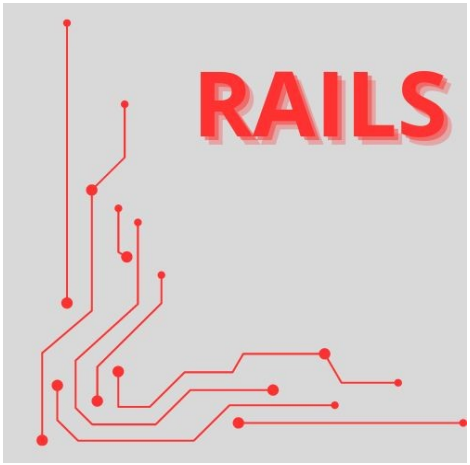
Introdução ao desenvolvimento de API's com

RAILS 7

Adyson Lima

Sumário



Capítulo 1

Qual o seu propósito ao ler este tutorial?

Como prática comum nos tutorias que escrevo, sempre tiro algumas linhas para perguntar ao leitor sobre o seu propósito. E neste tutorial, se o leitor me permitir, farei o mesmo, com o objetivo de motivar uma breve reflexão. Sendo assim ...

Qual o seu propósito ao ler este tutorial? Aprender uma nova habilidade para usar no seu trabalho? Passar o tempo lendo algo diferente? Ou quem sabe seu propósito seja algo mais profundo como por exemplo:

“ Meu propósito ao ler este tutorial é aprender uma nova habilidade e usar essa habilidade para resolver problemas reais no mercado de trabalho, ser respeitado e bem pago por isso. E além disso, dedicar 10% do meu ganho para ajudar pessoas em situação de morador de rua, fechando assim, meu ciclo de realização profissional ”.

Profundo hein? Brincadeiras a parte, o fato é que, **você deve ter um propósito claramente definido**, para impulsionar você para a **ação**. Pois a falta de um propósito, pode levar você a ter resultados abaixo do que você espera. Dito isso, sugiro que você pense alguns minutos sobre o seu propósito e responda a pergunta seguinte.

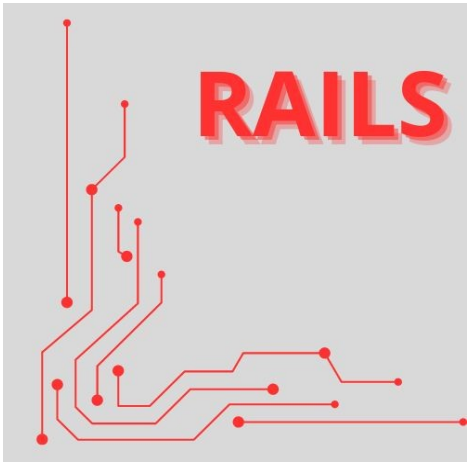
Qual o meu propósito ao ler este tutorial?

_____

_____

_____

Em seguida, avance para o próximo capítulo.



Capítulo 2

Uma palavra sobre vendas

Por que falar sobre vendas em um tutorial de programação? Bem, embora seja realmente incomum essa prática, sinto que é necessário lembrar o leitor que vendas ocorrem o tempo todo, por exemplo:

- *Vende-se ideias na internet.*

- *Vende-se pães na panificadora.*

- *Vende-se características pessoais e profissionais no Linkedin.*

- *Vende-se serviços.*

- *Vende-se produtos.*

E assim, é preciso lembrar que **softwares são vendidos** também, como API's por exemplo.

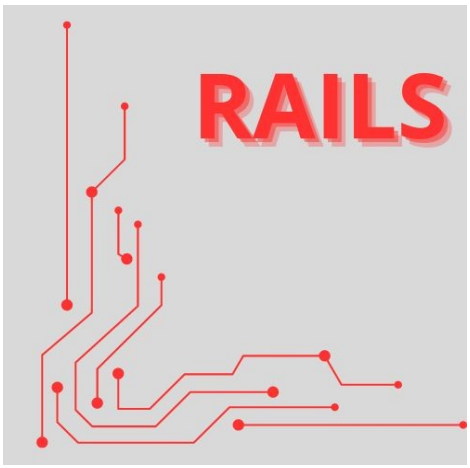
O leitor pode ter em mente que existem diversas maneiras de realizar vendas, de vendas no Instagram, até a clássica, mas ainda viva, venda através de panfletos.

Pense a respeito, você já pediu uma pizza ou outro prato de uma empresa que você tomou conhecimento através do Instagram ou de um panfleto?

Então, sugiro que o leitor busque aprender um pouco sobre vendas, existem excelentes materiais disponíveis, dê uma pesquisada no google, amazon, e você vai encontrar.

O investimento trará retorno com certeza. Pois **a habilidade de vender transcende áreas de conhecimento**, e uma vez aprendida, você poderá usá-la em diversas situações, de uma entrevista de emprego a uma transação contratual com um cliente. Além de ser uma profissão a mais na sua caixa de ferramentas.

E não fique surpreso de um dia ser contratado(a) como desenvolvedor, por alguém que leu um panfleto seu ou clicou em um anúncio seu no Instagram.

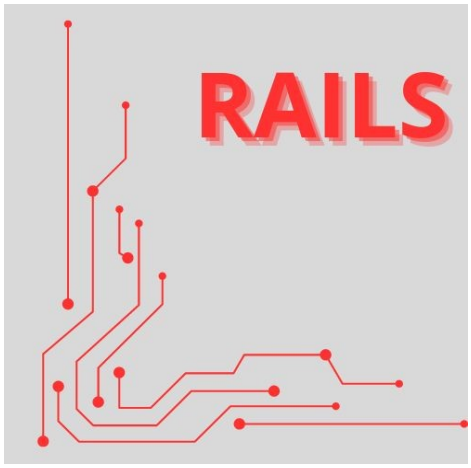


Capítulo 3

Para quem é destinado este material?

Este tutorial, é destinado principalmente a desenvolvedores Ruby, que queiram conhecer um meio de desenvolver Web API’s em Ruby on Rails, usando TDD. No entanto, pode ser lido por qualquer um que tenha interesse.

E mesmo desenvolvedores de software que usam outras tecnologias, podem ler este tutorial e assim, incrementar seus conhecimentos, e ainda ver como desenvolver em Ruby usando Rails pode ser gratificante.



Capítulo 4

Requisitos

Existem alguns requisitos necessários para tirar o melhor proveito deste tutorial, e creio que você seja desenvolvedor Ruby caro leitor e assim, você provavelmente já os tem. No entanto, segue a lista do que é necessário.

- Ruby 3 instalado.

- Rails 7 instalado.

- RSPEC instalado.

- VSCode instalado.

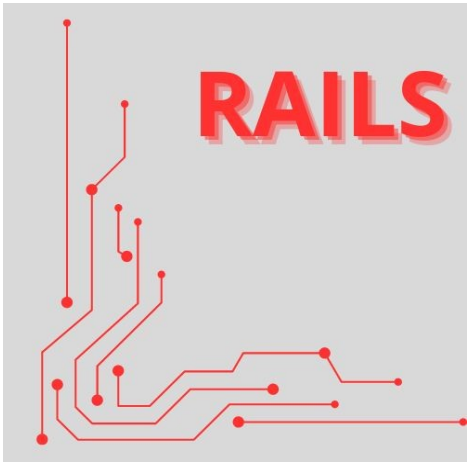
- Git instalado.

- Conta no GitHub.

- Conhecimento em lógica de programação.

- Conhecimento básico de desenvolvimento web.

Caso você não tenha os requisitos citados, você pode muito facilmente, encontrar de maneira abundante na internet fontes que podem suprir sua necessidade. Como por exemplo, o canal puts dev, do Xita.



Capítulo 5

Introdução

Desenvolver software é uma atividade que **demanda tempo**, e que **consome recursos mentais** do desenvolvedor. Nesse sentido, é muito importante ter ferramentas que reduzam o tempo de desenvolvimento e que consumam menos recursos mentais do programador. Por questões obvias, como, redução de **custo financeiro** para a empresa e menor desgaste da **saúde** do desenvolvedor.

Assim, surge este tutorial sobre Ruby on Rails, que é um framework Web super maduro, rico em recursos, seguro, estável e consagrado no mundo como uma das melhores opções para desenvolvimento Web.

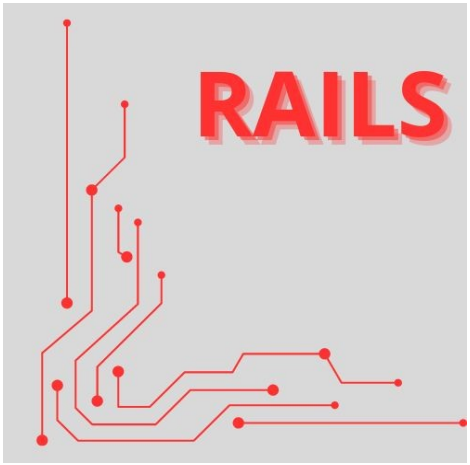
Este tutorial vai mostrar que é menos desgastante o processo de desenvolvimento usando Rails. Tanto pelas convenções usadas por padrão no Rails, que agilizam e reduzem muito o que é necessário configurar no projeto, tanto como pela quantidade de recursos disponíveis facilmente instaláveis.

E além disso, este tutorial irá mostrar como desenvolver usando testes no Rails. E sendo mais especifico, este tutorial vai abordar o TDD, já que esse é um padrão no mercado.

Uma revisão de Ruby será mostrada, com o foco no desenvolvimento com Rails. Assim como uma breve introdução a REST .

O desenvolvimento em Rails será no estilo mão na massa.

Então, sem mais delongas, vamos ao próximo capítulo.



Capítulo 6

Ruby

Ruby é uma linguagem de programação orientada a objetos, criada para ser fácil de usar, divertida e principalmente, poderosa. Objetivos que ela atende muito bem. Criada em 1995, por Yukihiro "Matz" Matsumoto, no Japão, é a base de desenvolvimento usada neste tutorial.

Assim, vamos para uma breve revisão sobre Ruby.

- Variáveis

A ideia de variáveis em Ruby, é semelhante a encontrada em muitas outras linguagens de programação, ou seja, variáveis são um espaço em memória onde é possível "guardar " uma informação de modo temporário. A sintaxe básica de uma variável é mostrada abaixo.

nome_da_variável = valor

Observe que em Ruby, não é utilizado ponto e virgula no final de instruções. E lembre que em Ruby, o tipo da variável é determinado pelo valor que é armazenado nela. Se o valor for inteiro, então a variável é do tipo inteiro, se o valor guardado for uma string, a variável é do tipo string e assim por diante. Vejamos mais alguns exemplos de uso de variáveis em Ruby.

idade = 20 # variável idade recebe um número

meu_nome = "josé" # variável meu_nome recebe um texto

motor_ligado = false # variável motor_ligado recebe um valor booleano

Em Ruby, existem também, variáveis de instância, que são variáveis que estão disponíveis para cada instância de um classe Ruby. Perceba o uso do arroba antes do nome desse tipo de variável. Veja um exemplo dessas variáveis usado no Rails.

@products = Products.all # variável de instância @products recebe todos os produtos

- Operações matemáticas

Realizar operações matemáticas é muito simples em Ruby, tal como os exemplos seguintes.

numero_um = 1 # variável numero_um recebe 1

numero_dois = 2 # variável numero_dois recebe 2

soma = numero_um + numero_dois # variável soma recebe a soma dos números

resto = numero_dois – numero_um # variável resto recebe a diferença dos números

produto = numero_um * numero_dois # variável produto recebe o produto dos números

quociente = numero_dois / numero_um # variável quociente recebe a divisão dos números

- Estruturas de decisão

Em Ruby, são usadas as estruturas **if**, **if else**, **elsif**, **unless**, **case**, **when**, e com exceção das três últimas, seu funcionamento é como em outras linguagens. Vejamos alguns exemplos simples de **if else**, em Ruby puro e dentro de uma aplicação Rails.

```
idade = 18 # variável idade recebe o número 18
if idade <= 17 # se a idade for menor ou igual a 17
  puts 'de menor' # mensagem se a condição if for verdadeira
elsif idade > 17 and idade < 65 # se a idade estiver entre 18 e 64 anos
  puts 'de maior' # mensagem se a condição elsif for verdadeira
else # caso as avaliações if e elsif não sejam verdadeiras
  puts 'senhor(a)' # mensagem para idade maior que 64 anos
end # fim do bloco
```

```
if @bread.update(bread_params) # se a função update executar coretamente
  render json: @bread #  renderize @bread como json
else # do contrário
  render json: @bread.errors, status: :unprocessable_entity # renderize os erros e o status
end # fim do bloco
```

- Loops

Executar loops em Ruby, especialmente dentro de aplicações Rails, que é nosso foco principal neste tutorial, é uma tarefa simples, mas um pouco diferente do usual em outras linguagens. Como podemos ver no exemplo abaixo em um arquivo **seeds.rb** do Rails, o uso de um loop que executa 5 vezes a instrução de criação de um registro no banco de dados. Note que não foi usado um laço **for**, como em outras linguagens de programação.

```
5.times do |i| # executa 5 vezes a criação de um objeto
  Bread.create(name: ['frances', 'hamburger', 'doce'].sample, price: 'preço tabelado')
end
```

- Classes

A declaração de classes, uma das bases da orientação a objetos, é feita em Ruby de acordo a sintaxe abaixo, correspondente a um controller e um model de uma aplicação Rails respectivamente.

```
class BreadsController < ApplicationController


end


class Bread < ApplicationRecord


end
```

Acima, foi declarado o model **Bread**, que é uma classe. E essa classe, herda de **ApplicationRecord**. O sinal de " < " indica a relação de herança.

- Objetos

Objetos em Ruby, como em muitas outras linguagens, são o recurso usável de uma classe, ou uma instância, se preferir. A sintaxe básica para criar um objeto em Ruby é mostrada abaixo. Observe o uso da função **new**.

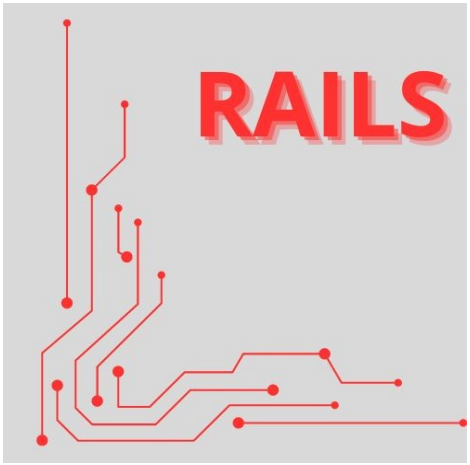
```
bread = Bread.new # objeto bread criado com a função new
```

Em Ruby, os objetos criados tem acesso aos recursos disponibilizados pela classe da qual são instanciados, como em outras linguagens.

- Juntando tudo em um pequeno programa Ruby

Agora, para finalizar essa breve seção de revisão sobre Ruby, vamos juntar as peças; Variáveis, estrutura if else, classes e objetos, em um pequeno programa. Os comentários dão uma explicação sobre o código de forma breve.

```
class Framework

 @name = ' ' # variável que armazena um nome de framework

 # função que salva um nome de um framework
 def save_name
  puts 'digite o nome de um Framework!'
  @name = gets.chomp
 end

 # função que imprime o nome de um framework
 def print_name
  if @name == 'Ruby on Rails'
   puts " #{@name} é baseado em Ruby! "
  else
   puts " #{@name} pode não ser baseado em Ruby! "
  end
 end

end

framework = Framework.new # instanciação da classe Framework

# chamada de duas funções da classe Framework
framework.save_name
framework.print_name
```



Capítulo 7

Ruby on Rails

Finalmente vamos falar sobre o framework para desenvolvimento web mais poderoso, e mais fácil de usar, Ruby on Rails, desculpe a empolgação :)

E para começar, vamos observar uma ideia presente no site “ https://rubyonrails.org/doctrine ”, escrita pelo DHH que diz muito sobre a ferramenta Ruby on Rails.

(*Provide sharp knives, ruby includes a lot of sharp knives in its drawer of features. Not by accident, but by design ...*)

Traduzindo,

(*Forneça facas afiadas, ruby inclui muitas facas afiadas em sua gaveta de recursos. Não por acidente, mas por design ...*)

Ruby, e Ruby on Rails trazem um conjunto de recursos extremamente úteis que agilizam e facilitam muito o trabalho de desenvolvimento. Como por exemplo, que tal um banco de dados pronto para uso, com distinção entre ambientes de teste, desenvolvimento e produção, e com uma ferramenta de ORM e um script para popular o banco rapidamente, incluídos por padrão com apenas um comando durante a criação de um projeto?

Bem, Ruby on Rails provê essa facilidade e creio que isso confirma a afirmação sobre “facas afiadas”.

E o que dizer sobre o excesso de poder presente no comando **scaffold** ? Para os que não conhecem, esse único comando cria um **crud** completo, tanto para aplicações fullstack quanto para api’s web.

E nem preciso dizer que caso você precise entregar algo muito rapidamente, você pode usar esse comando, **scaffold**. E associar a ele o uso do Bootstrap e fazer o deploy no Heroku. Claro que isso é para casos extremos, pois você não poderá fazer o ciclo TDD usando o **scaffold**, mas fica aí a ‘faca afiada’. “ Grandes poderes trazem grandes responsabilidades Peter ”, desculpe ;)

Agora, vamos partir para um pouco de prática com Ruby on Rails.

Tendo em mente que você já tem o Ruby e o Ruby on Rails instalados, abra um terminal no seu sistema, e digite o comando a seguir para criar um projeto.

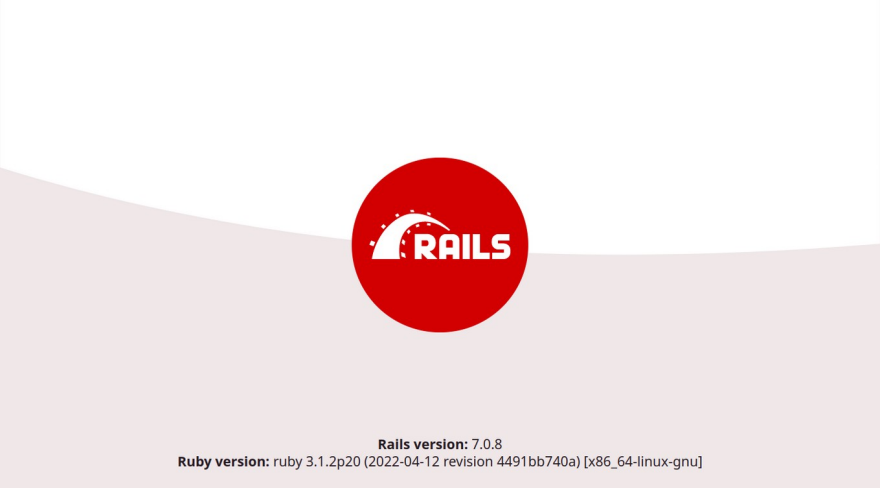
$ rails new meu_projeto --api --skip-test

Em seguida, entre na pasta do projeto e rode a aplicação com os comandos a seguir.

$ cd meu_projeto

$ rails s

Abra o navegador web e acesse **localhost:3000**, e você verá a tela de boa vindas do Rails, como na imagem a seguir.



Perceba o fato de que você precisou apenas de um comando para criar o projeto, e não precisou de várias configurações. Agora, vamos ver o que o comando **scaffold** oferece. Então, pare a aplicação pressionando a combinação de teclas **ctrl + c** no seu teclado, e digite os comandos abaixo.

$ rails g scaffold Car name age

$ rails db:migrate

Os dois comandos acima geraram pra você, uma api web, pronta para uso, que já pode ser testada usando o Postman, e que já atende requisições feitas por apps Android ou React por exemplo, bastando acessar os endpoints criados, lembra das ‘facas afiadas’ ?

O primeiro comando, **rails g scaffold Car name age**, gerou um Crud completo, com a estrutura da tabela **cars**, pronta para ser instalada no banco de dados, e incluiu nessa estrutura, os campos **name** e **age** para manipulação de dados.

O segundo comando, **rails db:migrate**, configurou o banco de dados e o deixou pronto para uso.

Agora, vamos configurar o arquivo **seeds.rb**, para gerar alguns registros no banco de dados e fazer alguns testes. Abra o arquivo **meu_projeto/db/seeds.rb** com seu editor de texto favorito e acrescente o código a seguir.

puts ‘gerando carros … ’

5.times do |i|

Car.create(name: [‘opala’, ‘maverick’, ‘dart’**].sample, age:** ‘mais de 30 anos’**)**

end

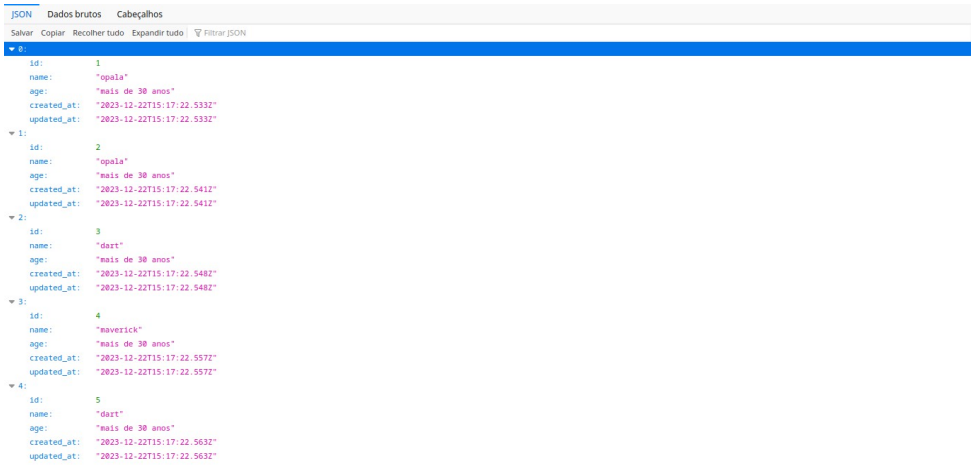
puts ‘carros gerados com sucesso!’

Agora, rode os comandos a seguir para executar as instruções do arquivo **seeds.rb**, e incluir 5 registros no banco de dados. E além disso, iniciar a aplicação novamente.

$ rails db:seed

$ rails s

Agora, acesse a aplicação no navegador web, ou no Postman, com o endereço **http://localhost:3000/cars**, e você verá a lista de carros gerada, como imagem a seguir.



Concluímos este capítulo agora, com uma pequena web api, desenvolvida com Rails. Vamos seguir para os próximos capítulos e ver um pouco sobre TDD e REST, para nos dar base para construirmos uma web api completa com Rails e TDD ao final deste tutorial.

Agradecimentos

Agradeço primeiramente a Deus. E agradeço aos meus pais que sempre me ajudaram. E ao leitor.

Referências

Beginning Ruby3, Carleton DiLeo, Peter Cooper, Fourth Edition

Ruby on Rails Tutorial, Michael Hartl, Fourth Edition