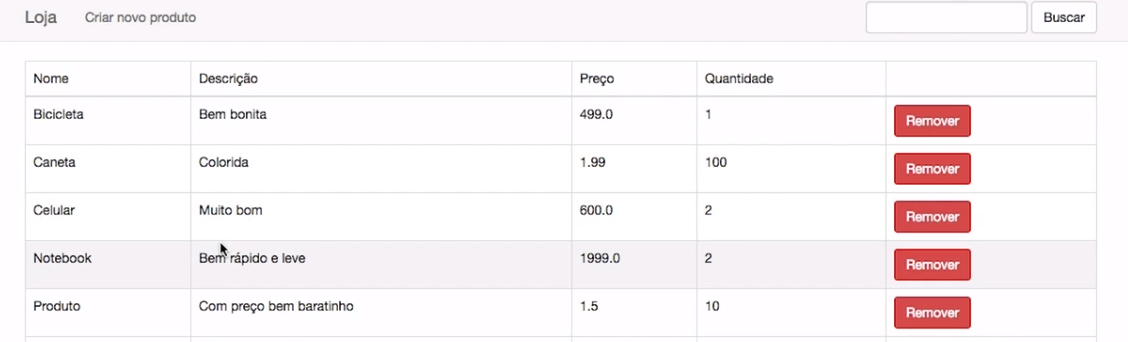
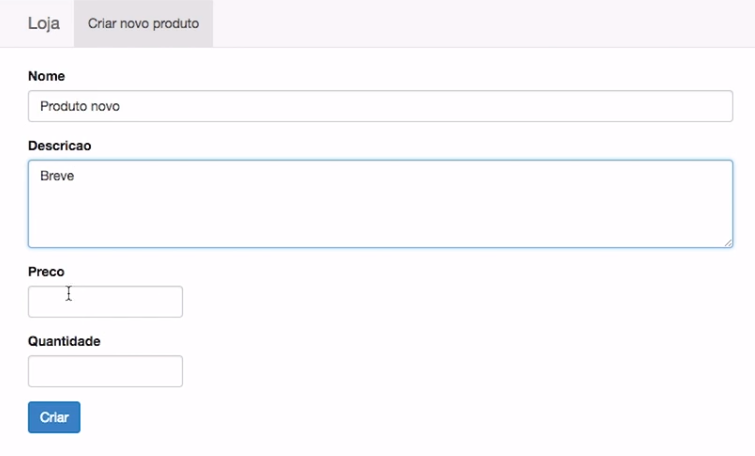
1. **Introdução**

**Transcrição**

Olá! Meu nome é Guilherme Lima, e neste curso de Rails 5 iremos lidar com a seguinte questão: temos uma uma loja on-line e queremos cadastrar todos os produtos disponíveis, de forma que o cliente possa visualizar quais são os itens disponíveis para venda. Teremos uma lista para mostrar todos os produtos, com a possibilidade de adicionar itens novos, bem como removê-los.



Na parte superior da tela, poderemos clicar sobre a opção "Criar Novo Produto", e seremos direcionados para um formulário que contém os campos "Nome", "Descrição", "Preço" e "Quantidade".



Ao criarmos um novo produto utilizando o formulário, ele estará visível na lista principal de produtos. Criaremos, ainda, uma categoria chamada "Produto com desconto", em que agruparemos os itens com menor preço. Como exemplo, criaremos um produto com o preço de 1.0, e ele será automaticamente agrupado na categoria de descontos, além de estar visível na lista geral de itens disponíveis.

Ao removermos esse produto novo, o navegador nos envia um pop-up perguntando se de fato queremos realizar a remoção. Ao clicarmos em "OK", outro produto com o preço menor em relação aos demais é agrupado na categoria "Produto com desconto", dessa forma teremos sempre em nossa loja algum item com desconto.

Criaremos também um campo de busca para que o usuário consiga encontrar itens com mais facilidade. Esse campo de busca ficará localizado na parte superior direita da tela. Uma vez que o cliente buscar um produto específico, serão exibidas as informações referentes a ele.

| **Nome** | **Descrição** | **Preço** | **Quantidade** |  |
| --- | --- | --- | --- | --- |
| Celular | Muito Bom | 600.0 | 2 | **Remover** |

Nesta primeira parte do curso, é essa estrutura que iremos montar. Vamos lá?

# 02 Preparando o ambiente

**Olá!**

Seja bem vindo ao treinamento de Ruby on Rails. Antes de iniciar nosso curso, vamos preparar o ambiente de desenvolvimento de acordo com seu sistema operacional.

**MAC**

* Homebrew nos permite instalar e compilar pacotes de software facilmente a partir da fonte. Para instalar, no terminal execute todo o comando com o link:

ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

* Para controlar a versão do ruby que vamos utilizar, no terminal execute o comando:

brew install rbenv ruby-build

* Em seguida, vamos escolher a versão do ruby com os seguintes comandos (execute um de cada vez):

rbenv install 2.5.1

rbenv global 2.5.1

* Agora, vamos instalar o rails com o seguinte comando:

gem install rails -v 5.2.0

* Para podermos usar o executável Rails, precisamos informar ao rbenv:

rbenv rehash

* Vamos instalar o banco de dados padrão do Rails, o SQLite com o seguinte comando:

brew install sqlite3

**Windows**

* [Instale o ruby clicando neste link com o kit de desenvolvimento](https://github.com/oneclick/rubyinstaller2/releases/download/rubyinstaller-2.5.3-1/rubyinstaller-devkit-2.5.3-1-x64.exe).
* Após o download, execute o instalador, aceite o termo de uso e prossiga com a instalação.
* Execute o comando para instalar o rails, na versão 5.2.

gem install rails -v 5.2.0

* Execute o comando abaixo para instalar o bundler.

gem install bundler

* Agora vamos precisar do Nodejs. Caso você já tenha instalado, maravilha. Caso ainda não tenha, [acessando esse link você pode fazer o download do Nodejs](https://nodejs.org/en/).
* Para finalizar a instalação, instale o banco de dados padrão do rails, o SQLite com o seguinte comando:

gem install sqlite3

**Linux**

* No terminal do linux, atualize a lista de programas que podem ser baixados com o seguinte comando:

sudo apt-get update

* Vamos instalar o curl para transferir os dados:

sudo apt-get install curl

gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E

\curl -L https://get.rvm.io | bash -s stable --ruby

* Agora vamos instalar o Nodejs com o seguinte comando:

sudo apt-get install nodejs

* Algumas gems são necessárias para o funcionamento do rails:

sudo apt-get install build-essential patch ruby-dev zlib1g-dev liblzma-dev

* Para finalizar, vamos instalar o rails:

gem install rails --no-document

* Vamos instalar também o git:

sudo apt-get install git

# 03 Primeira rota

## Transcrição

Temos vários produtos diferentes, e a ideia é criarmos uma estrutura de e-commerce simples para que os usuários consigam acessar a página e visualizar todos os itens disponíveis para compra. Cada produto deve possuir um nome, descrição breve, preço e quantidade.

Criaremos, portanto, uma aplicação utilizando o Ruby on Rails. Na atividade desta aula, foi disponibilizado um passo a passo de como preparar o ambiente de trabalho que iremos utilizar. Para começar, acessaremos o terminal - que usaremos bastante ao desenvolver um projeto utilizando Rails.

Para iniciarmos o projeto, temos um comando específico que ativará o Rails, mas antes de utilizá-lo, queremos que o diretório do projeto fique salvo na área de trabalho, tornando mais fácil a manipulação dos arquivos. Para isso, usaremos o comando cd destktop.

Last login: Mon Sep 17 10:01:06 on ttys000

Alura-Verde:~ aluraverde$ cd desktop

Alura-Verde:desktop aluraverde$

Em seguida, usaremos o comando rails new, para que o Rails crie um novo projeto para nós. Assim feito, devemos nomear o projeto a ser criado, como loja.

Last login: Mon Sep 17 10:01:06 on ttys000

Alura-Verde:~ aluraverde$ cd desktop

Alura-Verde:desktop aluraverde$ rails new loja

Vários arquivos serão criados e durante o curso entenderemos melhor a utilidade de cada um deles. Não trabalharemos exclusivamente com o terminal, faremos uso também do Visual Studio Code, mas se você quiser utilizar outro editor similar, como Sublime ou Atom, não há problema algum.

Abriremos o Visual Studio e arrastaremos a pasta do projeto para dentro do editor. Feito isso, teremos acesso a uma série de conteúdos criados pelo Rails, e um desses arquivos é muito importante, o Gemfile, onde encontraremos todas as gems que estamos utilizando no projeto. Temos, por exemplo, a versão do Ruby, do Rails, qual banco de dados será usado, assim como o web server.

source 'https://rubygems.org'

git\_source(:github) {|repo| "https://github.com/#{repo}.git" }

ruby '2.5.1'

# Bundle adge Rails instead: gem 'reails', github: 'rails/reails'

gem 'reails', '~> 5.2.1'

# Use sqlite3 as the database for Active Record

gem 'sqlite3'

# Use Puma as the app server

gem 'puma', '~> 3.11'

# Use SCSS for stylesheets

gem 'sass-rails', '~> 5.0'

Nós podemos acessar os arquivos criados pelo Rails e visualizar o projeto no navegador, para isso usaremos o comando cd loja para entrarmos no diretório do projeto, em seguida acionaremos o rails server:

Alura-Verde:loja aluraverde$ rails server

=> Booting Puma

=> Rails 5.2.1 application starting in development

=> Run `rails server -h` for more startup options

Puma starting in single mode...

\* Version 3.12.0 (ruby 2.5.1-p57), codename: Llamas in Pajamas

\* Min threads: 5, max threads: 5

\* Environment: development

\* Listening on tcp://0.0.0.0:3000

Use Ctrl-C to stop

Vimos que é a Puma que disponibiliza a aplicação para o servidor, e o listening é ouvido na porta tcp://0.0.0.0:3000. E esse endereço que utilizaremos no navegador para visualizarmos o projeto. Teremos uma mensagem de boas-vindas, mas não é esse o conteúdo que deve ser exibido em uma loja web, devemos mostrar os produtos disponíveis e suas características.



De volta ao Visual Studio, clicaremos no diretório app - que armazenará todos os códigos referentes à aplicação - em seguida clicaremos com o botão direito sobre a pasta views e selecionaremos a opção "New File". Esse novo arquivo, por padrão, será chamado de index.html.

Criaremos um código HTML simples, construindo uma tabela que conterá as colunas de Nome, Descrição, Preço e Quantidade.

<html>

<body>

<table>

<thread>

<tr>

<td>Nome</td>

<td>Descrição</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

</table>

</body>

</html>

Temos o cabeçalho pronto, resta criamos os produtos. O primeiro item será uma Camiseta, cuja descrição é Bem bonita, preço 30 e a quantidade, 10.

<html>

<body>

<table>

<thread>

<tr>

<td>Nome</td>

<td>Descrição</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

<tbody>

<tr>

<td>Camiseta</td>

<td>Bem bonita</td>

<td>30</td>

<td>10</td>

</tr>

</tbody>

</table>

</body>

</html>

O segundo produto será uma Luva, e preencheremos as características como descrição, preço e quantidade.

<html>

<body>

<table>

<thread>

<tr>

<td>Nome</td>

<td>Descrição</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

<tbody>

<tr>

<td>Camiseta</td>

<td>Bem bonita</td>

<td>30</td>

<td>10</td>

</tr>

<tr>

<td>Luva</td>

<td>Para os dias frios</td>

<td>39</td>

<td>5</td>

</tr>

</tbody>

</table>

</body>

</html>

Feito isso, poderemos acessar o navegador e visualizar os produtos na tabela que construímos. Digitaremos a URL localhost:3000/produtos. No entanto, teremos uma mensagem de erro do Rails:

Routing Error

No route matches [GET] "/produtos"

Rails.root: /Volumes/Alura Verde 2/aluraverde/Desktop/loja

Isso quer dizer que precisamos criar uma rota para conseguirmos acessar /produtos. Clicaremos sobre "Config > routes.rb", e nesse arquivo conseguiremos especificar quem atende a requisição enviada pelo navegador.

Rails.application.routes.draw do

# For details on the DSL avaible within this file, see

http://guides.rubyonrails.org/routing.html

end

Deletaremos os comentários sobre a documentação. Em seguida, especificaremos que quem deve atender o get "produtos" será produtos #index. O que estamos construindo é um **index de produtos**, para tornar nosso código mais organizado. Podemos construir um index de todas as categorias de produtos, cidades, inscrições, e assim por diante.

Rails.application.routes.draw do

get "produtos", to: "produtos#index"

end

Ao acessarmos a URL localhost:3000/produtos, teremos outra mensagem de erro:

Routing Error

uninitialized constant ProdutosController

Esse erro ocorreu porque toda a requisição necessita de um controle sobre o que deve ser exibido ou não, isto é, os dados que devem ser impressos na tela. Portanto, não basta apenas criarmos uma rota da requisição, precisamos também criar um elemento de controle. Esse erro ocorreu porque toda a requisição necessita de um controle sobre o que deve ser exibido ou não, isto é, os dados que devem ser impressos na tela. Portanto, não basta apenas criarmos uma rota da requisição, precisamos também de um **elemento de controle**.

O Rails nos ajudará nessa tarefa. No terminal, estamos executando o servidor. Ao acionarmos "Ctrl + C", o servidor será desligado e a página da loja não pode mais ser acessada no navegador. Para evitarmos esse problema, abriremos uma nova janela no terminal. Na primeira aba o servidor ficará no ar, e ele pode ser habilitado pelo comando rails s, e na outra janela escreveremos o código Ruby.

Primeiramente acessaremos nosso projeto, e em seguida usaremos o comando rails generate para gerar um controller de Produtos.

Alura-Verde:loja aluraverde$ rails generate controller Produtos

No Visual Studio, veremos que a nova pasta produtos é criada dentro de app. Essa nova pasta ainda está completamente vazia, e como havíamos criado um HTML para exibição de nossos produtos na pasta index.html, iremos movê-la para dentro de produtos. Ainda, será criada a classe produtos\_controller, localizada em "app > controllers". Essa classe herda algumas regras de ApplicationController que entenderemos posteriormente:

class ProdutosController < ApplicationController

end

De volta ao navegador, acessaremos o endereço localhost:3000/produtos. Não teremos mais a mensagem de erro, mas sim informações a respeito dos nossos dois produtos:

Nome Descrição Preço Quantidade Camiseta Bem bonita 30 10 Luva Para os dias frios 39 5

A ideia é cadastrarmos mais dois produtos. Já sabemos o procedimento: em index - agora armazenado na pasta produtos, criaremos mais tags <tr> e <td>. Trata-se de um processo bastante trabalhoso, e seria mais interessante que nossos produtos fossem cadastrados no banco de dados.

Vamos tornar essa ação possível nas próximas aulas.

## Transcrição

Nesta aula, a ideia é fazer o cadastramento dos produtos de nossa loja diretamente no banco de dados, e não no código HTML.

Primeiramente, precisamos ter um banco, e para isto acessaremos o terminal e solicitaremos sua criação pelo Rails por meio do comando rails db:create. Seremos notificados de que o banco de dados já existe e, por padrão, estamos usando o sqlite3.

Alura-Verde:loja aluraverde$ rails db:create

Database 'db/development.sqlite3' already exists

Alura:loja aluraverde$

O banco já existe, mas precisamos de fato utilizá-lo. Para isso, precisaremos de um modelo de produto. Sabemos que cada item possui nome, descrição, preço e quantidade, mas o banco de dados ainda não tem acesso a essas informações. O primeiro passo é gerar um modelo de produto, de forma que possamos gerar vários cadastros com a mesma estrutura.

No terminal, usaremos o comando rails generate para criar um model de Produto. Cada produto possui um nome, que é uma string; descricao do tipo text; preco do tipo decimal e quantidade, do tipo integer, isto é, um inteiro.

Assim feito, o arquivo db/migrate/20180917133141 será criado, e essa migração foi nomeada como create\_produtos.

Alura-Verde:loja aluraverde$ rails generate model Produto nome:string descica

o:text preco:decimal quantidade:integer

Running via Spring preloader in process 13111

invoke active\_record

create db/migrate/20180917133141\_create\_produtos.rb

create app/models/produto.rb

invoke test\_unit

create test/models/produto\_test.rb

create test/fixtures/produto.yml.

Alura-Verde:loja aluraverde$

Ao clicarmos na pasta "db > migrate", teremos o arquivo 20180917133141\_create\_produtos.rb criado:

class CrateProdutos < ActiveRecord::Migration[5.2]

def change

create\_table :produtos do |t|

t.string :nome

t.text :descricao

t.decimal :preco

t.integer :quantidade

t.timestamps

end

end

end

Foram criadas todas as informações que acompanham o cadastro de um produto, inclusive a t.timestamps, cuja função entenderemos mais adiante. Ainda não podemos visualizar os produtos no banco de dados, a migração apenas criou o arquivo 20180917133141\_create\_produtos, que deve ser passado para o banco por meio do comando rails db:migrate:

Alura-Verde:loja aluraverde$ rails db:migrate

== 20180917133141\_create\_produtos: migrating =============================================

=

-- crate\_table\_(:produtos)

-> 0.0026s

20180917133141 CreateProdutos: migrated (0.0028s) =====================================

=

Alura-Verde:loja aluraverde$

A migração de produtos foi criada, e também crate\_table\_(:produtos) na base de dados. Será que é possível visualizar os produtos na base de dados? Vamos limpar a tela do terminal usando o atalho "Command + L", ou "Ctrl + L". Para acessarmos a base de dados, usaremos o comando rails dbconsole:

Alura-Verde:loja aluraverde$ rails dbconsole

SQLite version 3.16.0 2016-11-04 19:09:39

Enter ".help" for usage hints.

sqlite>

Estamos na nossa base de dados, agora iremos visualizar as tabelas armazenadas no sqlite por meio do comando .table. Existem três delas: ar\_internal\_metadata,produtos e schema\_migrations.

Alura-Verde:loja aluraverde$ rails dbconsole

SQLite version 3.16.0 2016-11-04 19:09:39

Enter ".help" for usage hints.

sqlite> .table

ar\_internal\_metadata produtos schema\_migrations

splite>

Visualizaremos como está estruturada a tabela produtos, para isso usaremos os termos .schema produtos.

sqlite> .schema produtos

CREATE TABLE "produtos" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "nome"

varchar, "descricao" text, "preco" decimal, "quantidade" integer, "created\_

at" datetime NOT NULL, "updated+at" datetime NOT NULL);

sqlite>

É apresentado todo o esquema da tabela de produtos. O primeiro item se refere a um id, e embora não tenhamos especificado a criação desse elemento, o Rails produz automaticamente alguns campos a mais. Além do id teremos ainda created\_at e updated\_at, relacionados ao campo t.timestamps vistos no arquivo 20180917133141\_create\_produtos.rb.

Esses campos a mais são a infraestrutura do banco de dados do Rails, portanto todos os modelos que criarmos terá esses campos, sendo que os dois últimos envolvem quando o registro foi criado e sua última atualização.

Pressionaremos o atalho "Ctrl + C" para sair da base de dados, que ainda está vazia. Nós podemos inserir itens em nossa base utilizando o Rails, para tanto, usaremos o comando rails console. Nesse espaço podemos fazer testes e inserir produtos.

Alura-Verde:loja aluraverde$ rails console

Running via Spring preloader in process 13141

Loading development environment (Rails 5.2.1)

2.5.1 :001 >

Inseriremos uma camiseta, que equivale a Produto.new - utilizamos a letra maiúscula porque estamos nos referindo ao modelo. Serão apresentados os campos que criamos, ainda com valores nulos que precisam ser populados.

2.5.1 :001 > camiseta = Produto.new

=> #<Produto id: nil, nome: nil, descricao: nil, preco: nil, quantidade: nil,

created\_at: nil, updated\_at: nil>

Começaremos justamente a preencher esses campos. Portanto, camiseta.nome = "Camiseta azul", por exemplo. E seguiremos inserindo as informações sobre o produto. Por fim, salvaremos esse produto no banco de dados com a declaração camiseta.save:

2.5.1 :002 > camiseta.nome = "Camiseta azul"

=> "Camiseta azul"

2.5.1 :003 > camiseta.descricao = "Bem bonita"

=> "Bem bonita"

2.5.1 :004 > camiseta.preco = 19.99

=> 19.99

2.5.1 :005 > camiseta.quantidade = 10

=> 10

2.5.1 :006 > camiseta.save

Dessa forma, o Rails criará um novo produto:

(0.4ms) begin transaction

Produto Create (0.6ms) INSERT INTO "produtos" ("nome", "descricao", "preco", "quantidade", "created\_at", "updated\_at") VALUES (?,?,?,?,)

[["nome", "Camiseta azul"], ["descricao", "Bem bonita"], ["preco", 19.99], ["quantidade", 10], ["crate\_at", "2018-09-17" 13:38:00.580094"], ["updated\_at", ""2018-09-17 13:38:00.580094"]]

(1.5ms commit transaction)

Adicionaremos mais um produto, tenis. Contudo, existe outra forma de fazermos essa inserção de item, que é passando os parâmetros diretamente. Ao invés de escrevermos Produto.new, usaremos o Produto.create.

2.5.1 :007 > tenis = Produto.create nome:"Tenis", descricao:"Bom para corrida", preco:199, quantidade:3

Quando pressionamos o "Enter", o produto é inserido na base de dados. Será que se acessarmos o banco de dados em dbconsole, conseguiremos visualizar esses produtos cadastrados? Limparemos a tela do terminal por meio do atalho "Command + L" e acionaremos o comando rails bdconsole.

Alura-Verde:loja aluraverde$ rails dbconsole

SQLite version 3.16.0 2016-11-04 19:09:39

Enter ".help" for usage hints

sqlite>

Ao acessarmos o SQlite, utilizaremos dois comandos para que visualizemos o conteúdo em formato de tabela, sendo o primeiro .mode column, e o segundo, .hearder on. Por fim, acionaremos o select \* from produtos para visualizar o conteúdo de produtos:

Alura-Verde:loja aluraverde$ rails dbconsole

SQLite version 3.16.0 2016-11-04 19:09:39

Enter ".help" for usage hints

sqlite> select \* from produtos;

id nome descricao preco quantidade created\_at updated\_at

----- Camiseta azul Bem bonita 19.90 10 2018-09-17 2018-09-17 13:38:00.580094

Tenis Bom para c 199 3 2018-09-17 2018-09-17 13:38:00.580094

Os dois produtos foram criados na base de dados, os ids 1 e 2, que correspondem respectivamente aos itens Camiseta azul e Tenis. Já temos a base de dados criada utilizando o Rails console.

Nas próximas aulas, iremos alterar a nossa view para criarmos um código **HTML dinâmico**. Assim, o status dos produtos será relacionado ao banco de dados e às alterações realizadas.

# 05 Primeiros comandos

[Próxima Atividade](https://cursos.alura.com.br/course/ruby-on-rails-5/task/45867/next)

**Para criar nosso projeto usando Ruby on Rails, utilizamos o terminal para executar algumas ações e editamos o código no editor de texto de nossa preferência.**

**Sabendo disso, marque as alternativas verdadeiras.**

Top of Form

* Alternativa correta



O comando rails generate model ou rails g model seguido do nome do model que queremos criar, geram os mesmo resultado.

Bottom of Form

 Certo! Este comando gera uma classe que representará uma tabela do banco de dados da aplicação.

 Alternativa correta



O comando rails new cria toda a estrutura necessária para uma aplicação Rails.

 Certo! Além de criar tudo que precisamos para iniciar um novo projeto em Rails, não podemos esquecer de passar o nome do projeto, ou seja rails new loja.

 Alternativa correta



O comando rails generate controller gera um arquivo chamado routes.rb que cuida das rotas da aplicação.

 Na verdade esse comando gera uma classe capaz de atender a requisições feita por uma pessoa e não as rotas das requisições.

 Alternativa correta



O comando rails server ou rails s inicia nosso servidor.

Esse é o comando utilizado para subirmos nosso servidor local no endereço localhost:3000.

# 06 Criando um novo produto

**Uma pessoa estava tentando cadastrar seus produtos pelo rails console com o seguinte código:**

**produto = Produto.new**

**produto.nome = "Copo"**

**produto.descricao = "De vidro e bem bonito"**

**produto.preco = 20**

**produto.quantidade = 8**

**produto.save**

**Analisando o código acima, assinale as afirmações verdadeiras:**

Top of Form

* Alternativa correta



O produto não será cadastrado no banco de dados.

Bottom of Form

  Alternativa correta



O produto será cadastrado no banco de dados.

 Certo! O código acima não possui erros e será cadastrado no banco de dados.

 Alternativa correta



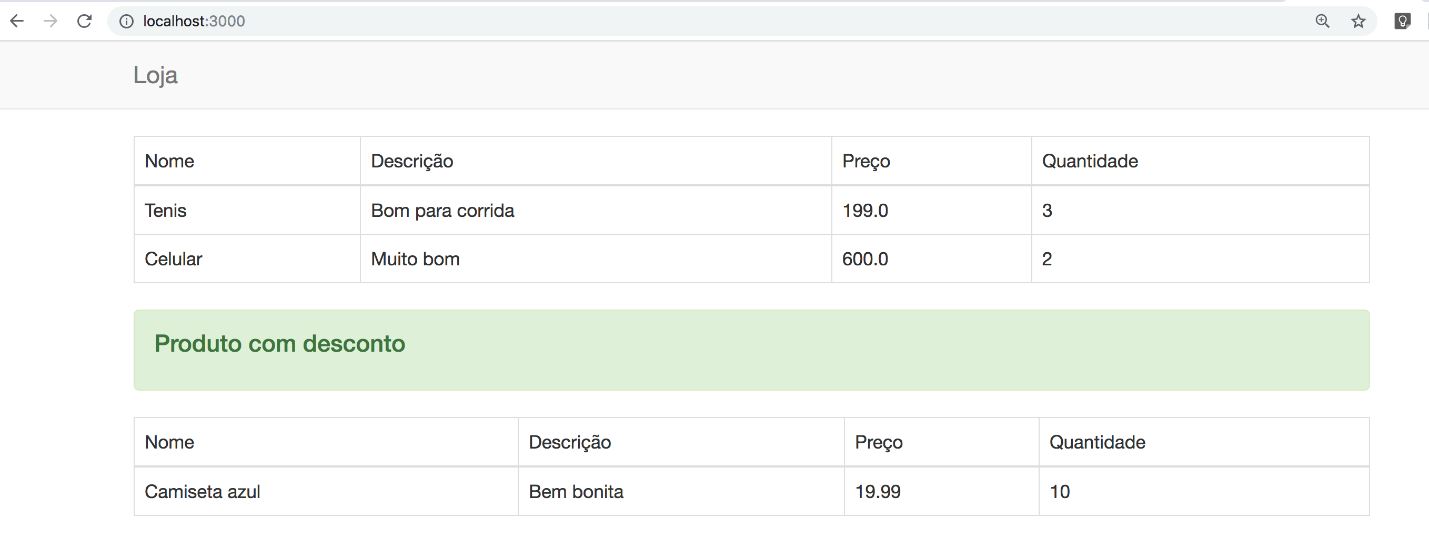
A única forma de cadastrar um produto é utilizar os métodos **new** e **save**.

## Nessa aula:

* Criamos um novo projeto chamado loja com o comando rails new loja
* Subimos o servidor na nossa máquina local para acessar nossa loja com o comando rails server ou rails s
* Criamos a [página](https://github.com/guilhermeonrails/alura_loja/blob/master/app/views/produtos/index.html) para listar os produtos da loja, incluímos uma rota e criamos um controle para acessar a página com o comando rails generate controller Produtos
* Geramos um modelo de produto para armazenar os dados no nosso banco de dados com o comando rails generate model produto nome:string descricao:text preco:decimal quantidade:integer
* Utilizamos o rails console ou rails c para criar os produtos no banco de dados

## Na próxima aula

Vamos ver como exibir os produtos que estão no banco de dados de forma dinâmica e melhorar a exibição da nossa página principal!



# 01 Exibindo produtos do banco

## Transcrição

Queremos trazer os produtos que temos da base de dados, e não utilizar um HTML estático como temos até então.

Para fazermos isso, devemos embutir código Ruby em nossa página HTML. Primeiramente, clicaremos com o botão direito sobre o arquivo index.html e o renomearemos para index.html.erb, de forma que essa página possa receber conteúdos em Ruby. Retiraremos o produto Luva do nosso código:

<html>

<body>

<table>

<thead>

<tr>

<td>Nome</td>

<td>Descrição</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

<tbody>

<tr>

<td>Camiseta</td>

<td>Bem bonita</td>

<td>30</td>

<td>10</td>

</tr>

</tbody>

</table>

</body>

</html>

Para conseguirmos escrever o código Ruby, utilizamos o sinal <% %>. O que faremos primeiro é trazer todos os produtos dentro de uma variável todos. Em seguida, por meio da variável, coletaremos produto por produto (each), e o chamaremos de forma individual de produto:

<tbody>

<% todos = Produto.all %>

<% todos.each do |produto| %>

<tr>

<td>Camiseta</td>

<td>Bem bonita</td>

<td>30</td>

<td>10</td>

</tr>

</tbody>

No caso do produto Camiseta, ao invés de ele ser exibido diretamente no código HTML, incluiremos Ruby e substituiremos Camiseta por produto.nome e o mesmo valerá para os outros campos. Por fim, fecharemos o loop each utilizando <% end %>.

<tbody>

<% todos = Produto.all %>

<% todos.each do |produto| %>

<tr>

<td><% produto.nome %></td>

<td><% produto.descricao %></td>

<td><% produto.preco %></td>

<td><% produto.quantidade %></td>

</tr>

<% end %>

</tbody>

A ideia é que os produtos sejam gerados dinamicamente. Vamos testar as modificações no navegador acessando a URL localhost:3030/produtos. Obteremos a seguinte mensagem na tela:

Nome Descrição Preço Quantidade

Nenhum produto foi exibido, pelo seguinte motivo: no Rails podemos apenas processar um código, portanto não precisamos processar todos os produtos. Como cada produto está sendo armazenado na variável todos, o que desejamos exibir é o conteúdo dos campos para o usuário, e para isso usaremos o sinal de =. Dessa forma essas informações estarão visíveis no navegador:

<tbody>

<% todos = Produto.all %>

<% todos.each do |produto| %>

<tr>

<td><%= produto.nome %></td>

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

</tr>

<% end %>

</tbody>

Ao atualizarmos a página no navegador, teremos os produtos e suas características impressas na tela.

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta azul | Bem bonita | 19,99 | 10 |
| Tenis | Bom para corrida | 199 | 3 |

Para garantir que esses são de fato os produtos cadastrados no banco de dados, abriremos o terminal e inseriremos mais um produto no rails console ou simplesmente rails c. Esse produto será um Celular, e terá todas as características de cadastro: descrição, preço e quantidade.

Alura-Verde:loja aluraverde$ rails c

Running via Spring preloader in process 13576

Loading development enviroment (Rails 5.2.1)

2.5.1 :001 > celular = Produto.create nome: "Celular", descricao:"Muito bom", preco:"600", quantidade:"2"

Para verificarmos se de fato o produto foi inserido, recarregaremos a página no navegador. Em seguida veremos que de fato o celular consta na tabela.

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta azul | Bem bonita | 19,99 | 10 |
| Tenis | Bom para corrida | 199 | 3 |
| Celular | Muito bom | 600 | 2 |

## Transcrição

Em nosso site, ainda bem simples, temos três produtos cadastrados:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Tenis | Bom para corrida | 199.0 | 3 |
| Celular | Muito Bom | 600.0 | 2 |

Eles não estão ordenados, e são exibidos de acordo com o momento de cadastramento de cada um deles, isto é, de forma aleatória, o que faz com que a experiência do usuário ao trafegar pela nossa lista de produtos não seja muito boa. É nisso que trabalharemos nesta aula.

Temos todos os produtos em nossa view index.html.erb:

<% todos = Produto.all %>

<% todos.each do |produto| %>

<tr>

<td><%= produto.nome %></td>

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

<tr>

Existe uma maneira de ordenarmos os produtos utilizando a palavra order ao invés de all, isto é, Produtos.order. Em seguida, especificaremos como deve se dar essa ordenação, no caso, guiada por nome.

<% todos = Produto.order :nome %>

<% todos.each do |produto| %>

<tr>

<td><%= produto.nome %></td>

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

<tr>

Ao atualizarmos o navegador, a tabela terá o seguinte aspecto:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Celular | Muito Bom | 600.00 | 2 |
| Tenis | Bom para corrida | 199.0 | 3 |

Podemos ordenar nossa tabela com base na descrição dos produtos, basta substituir nome por descricao. O mesmo vale para quantidade.

<% todos = Produto.order :descricao %>

<% todos.each do |produto| %>

<tr>

<td><%= produto.nome %></td>

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

<tr>

Ao ordenarmos nossos produtos, estamos modificando regras de negócio, pois acreditamos que a organização dos elementos pode melhorar a experiência do usuário em nosso site, gerando mais vendas e lucro. Contudo, estamos trabalhando com essa regra de negócio dentro da view index.hmtl.erb, o que faz muito sentido. O ideal é criarmos um local que armazene esse tipo de responsabilidade do projeto, e lá poderemos manipular as regras de negócio.

A criação de produtos\_controller.rb permite a possibilidade de dialogar com a view, desde que chamemos uma função com o mesmo nome deste, portanto na controller invocaremos a função index. Dessa forma, antes da view index ser acessada, primeiramente passaremos por produtos\_controller.rb, e dessa forma as configurações feitas serão aplicadas.

A função index que criamos exibirá todos os produtos por nome. Criaremos uma variável produtos que precisará ser acessada dentro da view, e para isso inseriremos um @, o que a transformará em uma variável de instância.

class ProdutosController < ApplicationController

def index

produtos = Produto.order : nome

end

end

De volta a index.html.erb, removeremos a linha <% todos = Produto.order : quantidade %> e no lugar da variável todos, passaremos @produtos:

<% produtos.each do |produto| %>

<tr>

<td><%= produto.nome %></td>

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

<tr>

Ao acessarmos o navegador, veremos a tabela ordenada por nome:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Celular | Muito Bom | 600.00 | 2 |
| Tenis | Bom para corrida | 199.0 | 3 |

Agora que sabemos que podemos dialogar com a view através da controller, poderemos configurar a visualização para que os produtos sejam ordenados de acordo com os seus preços.

class ProdutosController < ApplicationController

def index

produtos = Produto.order : nome

@produto\_com\_desconto = Produto.order : preco

end

end

Em index.hmtl.erb copiaremos a tabela inteira, e faremos uma segunda versão que será ordenada por preço. Usaremos a tag <h3> para marcar essa lista de produtos. Exibiremos, portanto, produtos\_com\_desconto em vez de apenas produtos.

<h3>Produto com desconto<h3>

<table>

<tr>

<td>Nome<td/>

<td>Descrição</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

<tbody>

<% @produtos\_com\_desconto.each do |produto| %>

<tr>

<td><%= produto.nome %></td

<td><%= produto.descricao %></td>

<td><%= produto.preco %></td>

<td><%= produto.quantidade %></td>

</tr>

# código omitido

</table>

Dessa forma, no arquivo index.html.erb teremos duas tabelas: uma dos produtos cadastrados como um todo, e uma segunda ordenada por preço, que destaca os itens promocionais. Ao acessarmos nosso site no navegador por meio da URL localhost:3000/produtos, teremos duas tabelas diferentes:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Celular | Muito Bom | 600.00 | 2 |
| Tenis | Bom para corrida | 199.0 | 3 |

\*\*Produto com desconto

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Tenis | Bom para corrida | 199.0 | 3 |
| Celular | Muito bom | 600.0 | 2 |

Teremos todos os produtos cadastrados sendo exibidos e ordenados por preço. E como queremos exibir apenas um único produto com desconto, em produtos\_controller envolveremos preco com () e estipularemos um limite de apenas 1 produto. Então será buscado o produto ordenado por preço e exibido apenas o primeiro.

class ProdutosController < ApplicationController

def index

produtos = Produto.order : nome

@produto\_com\_desconto = Produto.order(:preco).limit 1

end

end

Ao acessarmos a página no navegador, teremos apenas um produto com desconto na tabela:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |
| Celular | Muito Bom | 600.00 | 2 |
| Tenis | Bom para corrida | 199.0 | 3 |

\*\*Produto com desconto

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |

O produto exibido na sessão de descontos é "Camiseta Azul", mas repare que ela também é visível na tabela superior como primeiro item. Queremos que o produto com desconto seja mostrado apenas na sessão de descontos. Para isso, ordenaremos nome da primeira tabela de forma descendente (desc), e não ascendente como é o padrão. Feito isso, configuraremos um limite de exibição de 2 nomes.

class ProdutosController < ApplicationController

def index

produtos = Produto.order(nome: :desc).limit 2

@produto\_com\_desconto = Produto.order(:preco).limit 1

end

end

Desse modo, ao serem acessadas via navegador, as tabelas terão a seguinte configuração:

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Celular | Muito Bom | 600.00 | 2 |
| Tenis | Bom para corrida | 199.0 | 3 |

\*\*Produto com desconto

| **Nome** | **Descrição** | **Preço** | **Quantidade** |
| --- | --- | --- | --- |
| Camiseta Azul | Bem bonita | 19.99 | 10 |

Conseguimos isolar os produtos de acordo com a nossa regra de negócio, de forma que os produtos com desconto são exibidos em uma tabela diferente.

O próximo passo é refletir sobe a experiência do usuário ao acessar nosso site. Se o endereço da URL digitado for localhost:3000, será exibido na tela a mensagem de boas-vidas do Rails. O ideal seria que esse endereço levasse o usuário para localhost:3000/produtos, ou seja, para as tabelas de produtos que acabamos de construir.

Por lidarmos com as rotas da aplicação, selecionaremos "config > routes.rb":

Rails.application.routes.draw do

get "produtos", to: "produtos#index"

end

Construiremos uma rota nesse arquivo, primeiramente substituindo o termo get por root to, que apontará para produtos#index:

Rails.application.routes.draw do

root to: "protudos#index"

end

Agora, se acessarmos localhost:3000/produtos, receberemos uma mensagem de erro, "Routing Error", que indica que não há quem atenda tal requisição. Para acessarmos nosso site, o novo endereço é localhost:3000, a ser atendido por produtos\_controller, e assim o método index acionará a view de mesmo nome e as tabelas serão exibidas.

A próxima fase do nosso projeto é melhorar o aspecto visual do site, vamos lá!

## Transcrição

Aprendemos a modificar a rota principal de acesso para as tabelas de produtos. Anteriormente, a via de acesso era o endereço localhost:3000/produtos. Em "Config > routes.rb" tínhamos uma requisição get "produtos", que verifica se há um método index operando, e em seguida aponta para a view de mesmo nome.

Para modificarmos a rota principal utilizamos o termo root to:

Rails.application.routes.draw do

root to: "produtos#index"

end

Desse modo, só poderemos acessar as tabelas de produtos por meio de localhost:3000. Caso ainda tentemos utilizar localhost:3000/produtos, teremos uma mensagem de erro de rota. Podemos visualizar quais são as rotas disponíveis nos projetos no próprio navegador, desde que o servidor esteja no ar. Para isso, acessaremos localhost:3000/reails/info/routes.

Nessa página, podemos visualizar todas as rotas disponíveis do projeto. Teremos, por exemplo, o /, que é atendido pela URL produtos#index. Os outros arquivos servem para uso interno no Rails, e não nos atentaremos para estes pormenores ainda.

| **Helper** | **Http Verb** | **Path** | **Controller#Action** |
| --- | --- | --- | --- |
| root\_path | GET | / | produtos#index |
| rails\_service\_blob\_path | GET | /rails/active\_storage/blobs/:signed\_id/\*filename(.:format) | active\_storage/blobs#show |
| rails\_blob\_representation\_path | GET | /rails/active\_storage/representations/signed blobid/:variation\_key/\*filename(:.format) | active\_storage/representations#show |
| rails\_disk\_service\_path | GET | /rails/active\_storage/disk/encoded\_key/\*filename(:format) | active\_storage/disk#show |
| update\_rails\_disk\_service\_path | GET | /rails/active\_storage/disk/:encoded\_token(.:format) | active\_storage/disk#update |
| rails\_direct\_upload\_path | PUT | /rails/active\_storage/direct\_uploads(.:format) | active\_storage/direct\_uploads#create |

Já conseguimos acessar nosso site pelo endereço localhost:3000, mas o aspecto visual do site deixa a desejar. Poderemos melhorar essa parte do projeto utilizando **Bootstrap**, um framework de código aberto voltado para o desenvolvimento front end, para deixarmos o visual do nosso site mais agradável para o usuário.

Caso não estivéssemos utilizando o Rails, teríamos de realizar o download das pastas do Bootstrap e incluir os arquivos CSS e JS do framework nos arquivos do nosso projeto. Utilizamos o Rails para incorporar as bibliotecas Bootstrap de forma mais simples.

Acessaremos o link do [Twitter Bootstrap Rails](https://github.com/seyhunak/twitter-bootstrap-rails), em que encontraremos uma breve documentação que nos orienta em como realizar a instalação. O primeiro passo é trazer a gem twitter-bootstrap-rails para o projeto.

Acessaremos o arquivo Gemfile, que contém todas as bibliotecas que estamos utilizando, e criaremos a nova gem especificada.

source 'https://rubygems.org'

git\_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '2.5.1'

gem 'twitter-bootstrap-rails'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'

gem 'rails', '~> 5.2.1'

# Use sqlite3 as the database for Active Record

gem 'sqlite3'

# Use Puma as the app server

gem 'puma', '~> 3.11'

# Use SCSS for stylesheets

gem 'sass-rails', '~> 5.0'

# Use Uglifier as compressor for JavaScript assets

gem 'uglifier', '>= 1.3.0'

Não basta apenas escrevermos essa gem no arquivo, a própria documentação deixa isso claro, e solicita que depois executemos o comando bundle install. Esse comando verificará se todas as dependências do projeto já estão presentes.

No terminal, acessaremos o diretório loja e acionaremos o comando:

Last login: Mon Sep 17 13:10:46 on ttys00

Alura-Verde:loja aluraverde$ bundle install

Por fim, a nova gem já está operante. Nós não precisamos estilizar o Bootstrap, podemos aproveitar o formato padrão. Usaremos o comando rails generate bootstrap: instal static no terminal, e com isto alguns arquivos serão criados para que o framework possa ser utilizado no projeto.

Alura-Verde:loja alura verde$ rails generate bootstrap:install static

Running via Spring poreloar in process 14683

Feito isso, ao acessarmos localhost:3000 teremos uma mensagem de erro: Sprockets::FileNotFound in Produtos#index. Por que isso aconteceu?

Algumas gems que utilizamos necessitam que recarreguemos o servidor. No terminal, utilizaremos o atalho "Ctrl + C" para interrompê-lo. Em seguida, subiremos novamente o servidor por meio do comando rails s.

Já no navegador, atualizaremos a página e veremos que não há mais a mensagem de erro. Teremos algumas alterações nas tabelas, que ainda não são modificações estéticas satisfatórias.

Existem duas classes do Bootstrap que podemos utilizar em nossas tabelas - você também pode pesquisar outros recursos na documentação desse framework, disponível em seu [site oficial](https://getbootstrap.com/) .

Importaremos duas classes para o arquivo index.html.erb, em <table>, table table-bordered e table-hover. Faremos o mesmo precedimento tanto para a tabela de produtos cadastrados quanto para a de descontos.

<html>

<body>

<table class="table table-bordered table-hover">

<thead>

<tr>

<td>Nome</td>

<td>Descricao</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

# código omitido

</body>

</html>

Teremos um aspecto visual mais agradável, mas ainda existem vários pontos a serem melhorados. O usuário entra em contato com as tabelas de forma muito crua - seria interessante criarmos uma espécie de menu de navegação. No site da **Alura**, por exemplo, temos um menu na parte superior da tela que redireciona o usuário para os diversos setores da página, como "Todos os Cursos", "Carreiras e Formações", e assim por diante, além do brand name da empresa, isto é, a logomarca associada a um link.



Na documentação do Twitter Bootstrap, no tópico "Navbar", encontraremos uma barra de navegação chamada "Brand name", mesmo conceito que vimos no site da **Alura**. O código para implementar essa barra na navegação é:

<%= nav\_bar brand: "We're sooo web 1.0alizr", brand\_link: account\_daskboard\_path %>

Em index.html.erb, criaremos a navbar entre as tags <html> e <body>. Daremos o nome da brand de Loja, e o link apontado será root\_url, ou seja, a raiz do nosso projeto.

<%= nav\_bar brand: "Loja", brand\_link: root\_url %>

<body>

<table class="table table-bordered table-hover">

<thead>

<tr>

<td>Nome</td>

<td>Descricao</td>

<td>Preço</td>

<td>Quantidade</td>

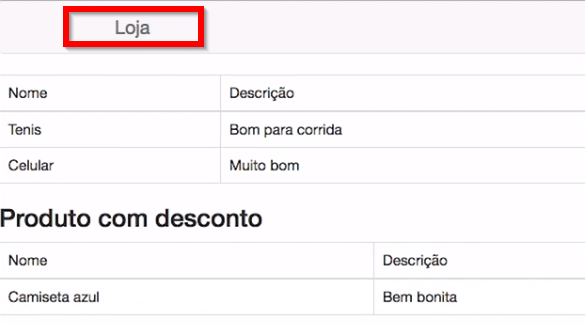
</tr>

</thead>

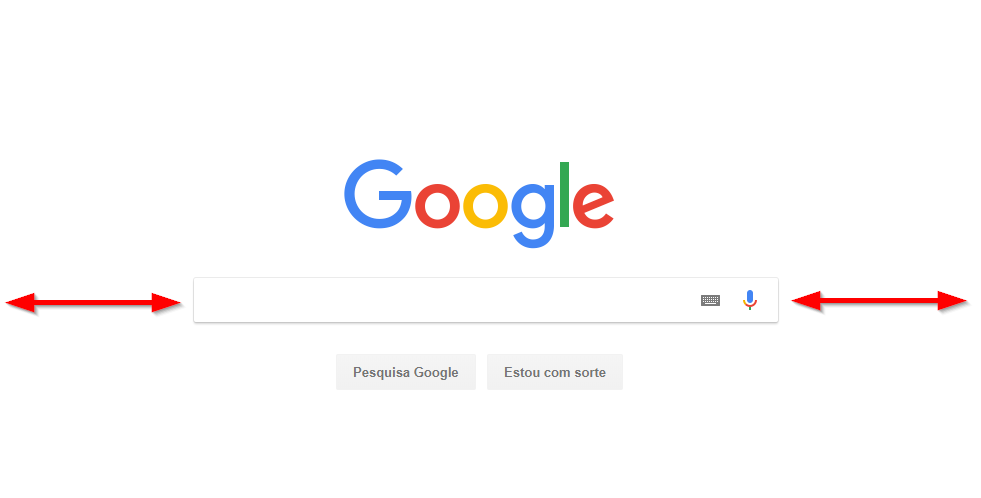
# código omitido

</body>

Ao atualizarmos a página no navegador, já teremos o nome da marca disponível no menu de navegação. Ao clicarmos sobre ela, somos direcionados para a mesma página, a única que possuímos no projeto.



Perceba que a tabela exibida se estende do lado esquerdo até o extremo direito da tela. Teríamos um aspecto visual mais interessante se as nossas tabelas fossem centralizadas, criando um intervalo entre as extremidades da tela, tal qual existe no site do buscador Google, por exemplo.



Antes do corpo do HTML (<body>), criaremos uma <div>, que receberá uma classe do Bootstrap chamada container, lembrando que precisamos fechar essa <div> no final do código.

<html>

<%= nav\_bar brand: "Loja", brand\_link: root\_url %>

<div class="container">

<body>

<table class="table table-bordered table-hover">

<thead>

<tr>

<td>Nome</td>

<td>Descricao</td>

<td>Preço</td>

<td>Quantidade</td>

</tr>

</thead>

# código omitido

</html>

Ao atualizarmos a página, veremos as tabelas centralizadas. Por fim, adicionaremos maior destaque na frase "Produtos com desconto". Incluiremos uma classe do Bootstrap associada a uma nova <div>, a alert alert-success, que receberá a função role="alert". Dentro das <h3> que envolvem a mensagem, incluiremos a segunda classe alert-heading. Por fim, fecharemos a <div>.

<div class="alert alert-success" role="alert">

<h3 class="alert-heading">Produto com desconto</h3>

</div>

<table class="table table-bordered table-hover">

Ao atualizarmos a página, veremos as modificações aplicadas. O visual do site já está muito mais agradável.



Podemos, ainda, modificar o header do título "Produtos com desconto" para conseguirmos um resultado mais satisfatório, trocaremos as <h3> por <h4>:

<div class="alert alert-success" role="alert">

<h4 class="alert-heading">Produto com desconto</h4>

</div>

<table class="table table-bordered table-hover">

Dessa forma, nosso site está com um destaque, e tabelas centralizadas, melhorando a experiência do usuário.

# 04 Ordenação

[Próxima Atividade](https://cursos.alura.com.br/course/ruby-on-rails-5/task/45944/next)

**Dica: Começando daqui ? -** [**Para visualizar ou fazer download do projeto da aula anterior, clique neste link**](https://github.com/alura-cursos/ruby-on-rails-5/archive/master.zip)

**A figura abaixo ilustra a tabela de produtos de uma pessoa:**

| id | **nome** | **descricao** | **preco** | **quantidade** |
| --- | --- | --- | --- | --- |
| 1 | Blusa | Azul | 199.99 | 2 |
| 2 | Livro | Aventura | 29.99 | 5 |
| 3 | Camiseta | Amarela | 50 | 10 |
| 4 | Luva | Cinza | 39.90 | 1 |

**Para melhorar a forma que os produtos serão exibidos na web, a pessoa testou:**

**1 - ordenar por quantidade de forma decrescente**

**2 - ordenar apenas os 3 produtos com preço mais alto**

**Sabendo disso, selecione as afirmações verdadeiras:**

Top of Form

* Alternativa correta



O código usado para a ordenação do teste **2** foi:

@produtos = Produto.order(preco: :desc)

Bottom of Form

  Alternativa correta



O código usado para a ordenação do teste **2** foi:

@produtos = Produto.order(preco: :desc).limit 3

 Certo! Podemos limitar o resultado com o código limit, seguido do valor limite.

 Alternativa correta



O código usado para a ordenação do teste **1** foi:

@produtos = Produto.order(quantidade: :desc)

 Certo! Utilizamos o símbolo :desc para indicar uma ordenação decrescente.

 Alternativa correta



O código usado para a ordenação do teste **1** foi:

@produtos = Produto.order(quantidade: :asc)

# 05 Processar ou exibir?

[Próxima Atividade](https://cursos.alura.com.br/course/ruby-on-rails-5/task/45868/next)

**Após incluir os produtos no nosso banco de dados utilizando o rails console, alteramos o arquivo index.html para index.html.erb para incluir código ruby e exibir nossos produtos de forma dinâmica.**

**Porém, em alguns momentos precisamos apenas processar código sem exibir na tela e em outros momentos exibir na tela.**

**Sabendo disso, marque as opções verdadeira:**

Top of Form

* Alternativa correta



Quando queremos processar código ruby sem exibir na tela, utilizamos a sintaxe **<%** e **%>**.

Bottom of Form

 Certo! Para uma página aceitar código Ruby, ela deve estar entre **<%** e **%>**.

 Alternativa correta



Quando queremos exibir código ruby na tela, utilizamos o símbolo **<%=** e **%>**.

 Certo! Com a sintaxe **<%=**, executamos o código e imprimimos na página HTML.

 Alternativa correta



Quando queremos processar e exibir código ruby, utilizamos a sintaxe **<%** e **%>**.

# 06 O que vai acontecer?

[Próxima Atividade](https://cursos.alura.com.br/course/ruby-on-rails-5/task/46009/next)

**Certa pessoa decidiu buscar os produtos que inseriu utilizando o rails console, alterando o código da index de produto para:**

**<% todos\_produtos = Produto.all %>**

**<% todos\_produtos.each do |produto| %>**

**<tr>**

**<td> <% produto.nome %> </td>**

**<td> <% produto.descricao %> </td>**

**<td> <% produto.preco %> </td>**

**<td> <% produto.quantidade %> </td>**

**</tr>**

**<% end %>**

**Analisando o código acima, podemos afirmar que:**

Top of Form

* Alternativa correta



Todos os produtos do banco de dados serão exibidos, já que o código não possui erros.

Bottom of Form

  Alternativa correta



Nenhum produto será exibido e uma mensagem de erro será mostrada no navegador.

  Alternativa correta



Nenhum produto será exibido e nenhuma mensagem de erro será mostrada no navegador.

Certo! No código acima existe um erro de sintaxe, já que foi utilizado o símbolo **<%**, que processa código ruby, porém não exibe na view. Quando queremos exibir na view, é necessário o uso do símbolo **<%=**.

# O que aprendemos?

[Próxima Atividade](https://cursos.alura.com.br/course/ruby-on-rails-5/task/46010/next)

## Nessa aula:

* Exibimos a lista de produtos que inserimos no banco através do rails console na index de produtos
* Embutimos código ruby na página alterando a extensão da página de index**.html** para index**.html.erb**
* Ordenamos a lista de produtos por nome e pelo produto com menos valor
* Adicionamos o bootstrap no nosso projeto, para ter um visual mais agradável

## Na próxima aula

Vamos entender melhor a arquitetura MVC e criar uma interface para cadastrar novos produtos!

