

# Ruby on Rails 5.x

*Do início ao fim!*

# Iniciando o módulo 3

## Iniciando o módulo 3

- Vamos desenvolver nossa primeira aplicação.
- A aplicação será simples.
- A importância desse módulo vai ser conhecer alguns conceitos importantes sobre o Rails.
- Vamos conhecer inicialmente onde buscar ajuda. Documentação!

# Documentação

# Ruby

<https://www.ruby-lang.org/pt/>

# Ruby

- Documentação
  - <https://www.ruby-lang.org/pt/documentation/>
- Core
  - <http://ruby-doc.org/core-2.5.1/>
- Standard Library
  - <http://ruby-doc.org/stdlib-2.5.1/>
- API Dock
  - <https://apidock.com/ruby>

# Ruby

- Documentação
  - <https://www.ruby-lang.org/pt/documentation/>
- Core
  - <http://ruby-doc.org/core-2.5.1/>
- Standard Library
  - <http://ruby-doc.org/stdlib-2.5.1/>
- API Dock
  - <https://apidock.com/ruby>

# Ruby on Rails

<https://rubyonrails.org/>

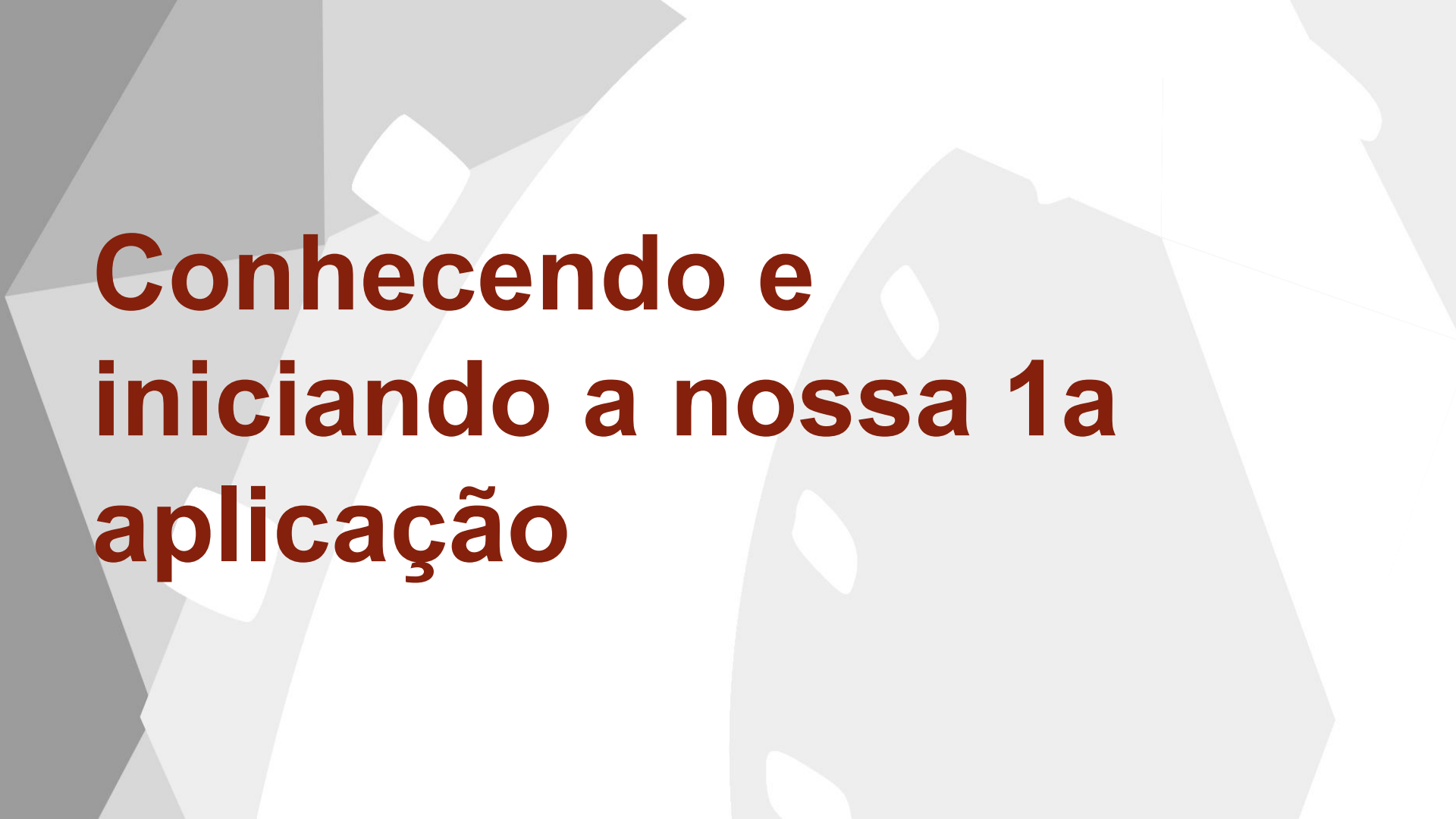


# Ruby on Rails

- Guides
  - <http://guides.rubyonrails.org/>
- API
  - <http://api.rubyonrails.org/>
- API Dock
  - <https://apidock.com/rails>

# Documentação Offline

<http://devdocs.io/>



# **Conhecendo e iniciando a nossa 1a aplicação**



**Antes...**

# Newsletter Semanal

[www.videosdeti.com.br/newsletter](http://www.videosdeti.com.br/newsletter)



# Crypto Wallet

# Crypto Wallet

1a Parte

# Crypto Wallet

- Cadastrar as moedas, suas imagens e siglas

- Ex:

- Bitcoin - BTC
    - Litecoin - LTC
    - Dash - DASH
    - Ethereum - ETH





# Qual “stack” vamos usar?

ruby (linguagem)

ruby on rails (framework)

Sqlite (banco de dados)

# Gere a aplicação

Use a versão **5.2** do Rails

```
rails _5.2_ new crypto_wallet
```

# Veja outras opções

Digite `"rails"` e dê ENTER

# **Usando outro BD e Conhecendo as pastas do projeto**



**Usando outro BD**

## Usando outro BD

- Para saber qual o BD que está sendo usado na aplicação, verifique o arquivo ***config/database.yml***
- Digitando ***rails*** fora da pasta podemos ver a opção ***--database*** que permite a configuração para outros BD.
- Ex:
  - `rails new meu_app --database=<nome bd>`

# **Conhecendo as pastas do projeto**

# Conhecendo as pastas do projeto

- Você pode verificar a funcionalidade de cada pasta em:
  - `http://guides.rubyonrails.org/v5.2/getting\_started.html#creating-the-blog-application`



The background features a complex arrangement of overlapping geometric shapes in various shades of gray. A prominent light gray film strip with several white rectangular sprocket holes runs diagonally across the frame. The word "Prototipando" is centered in a bold, dark red font.

# Prototipando

# Prototipando

- Quando falamos em protótipo, estamos falando em tentar passar, na maioria das vezes para o cliente, como o software ficará visualmente, permitindo que o cliente possa entender e opinar sobre os aspectos visuais e funcionais do sistema.

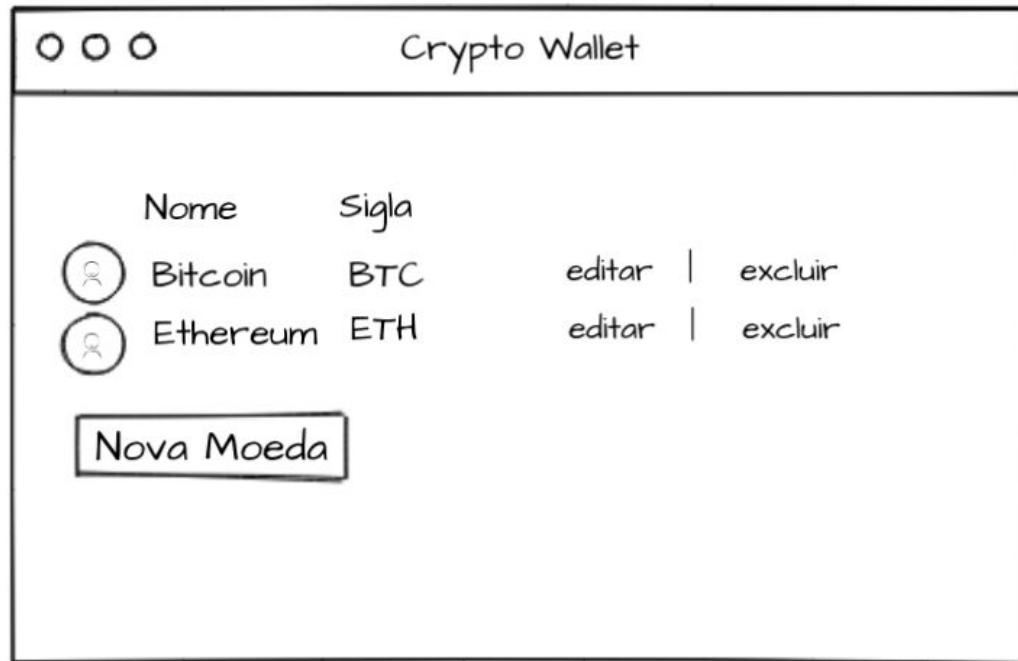
# Prototipando

- A prototipagem pode ter vários níveis de fidelidade, desde o mais baixo (wireframes) até o mais alto com muitos níveis de detalhes dando a impressão do software funcionando.

# Prototipando

- Algumas ferramentas ajudam a prototipar e também a gerar código dando vida ao protótipo. Veja esses exemplos:
  - **Adobe XD**  
(<https://www.adobe.com/br/products/xd.html>)
  - **Balsamiq** (<https://balsamiq.com/>)
  - **Mock Flow** (<https://mockflow.com>)

# Prototipando



# Modelagem de Dados

The background of the slide features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

# Modelagem de Dados



# Modelagem de Dados

- A Modelagem de Dados é a criação de um modelo físico que explique a lógica por trás do sistema, com ele você é capaz de explicar as características de funcionamento e comportamento de um software.
- A modelagem de dados é a base de criação do Banco de dados e parte essencial para a qualidade do software



# Modelagem de Dados



- Imagine uma tabela de excel...

Bitcoin	BTC	
Litecoin	LTC	

# Modelagem de Dados

- Imagine uma tabela de excel...

Moedas

Bitcoin	BTC	
Litecoin	LTC	

# Modelagem de Dados

- Imagine uma tabela de excel...

Moedas

Bitcoin	BTC	<a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>
Litecoin	LTC	<a href="http://site.com/litecoin.jpg">http://site.com/litecoin.jpg</a>

# Modelagem de Dados

- Imagine uma tabela de excel...

## Moedas

Nome	Sigla	Imagem
Bitcoin	BTC	<a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>
Litecoin	LTC	<a href="http://site.com/litecoin.jpg">http://site.com/litecoin.jpg</a>

Campo

Registro

The background features a light gray film strip with white sprocket holes, winding across the frame. It is overlaid with various geometric shapes in shades of gray, including a large triangle on the left and a large circle on the right.

**MVC**



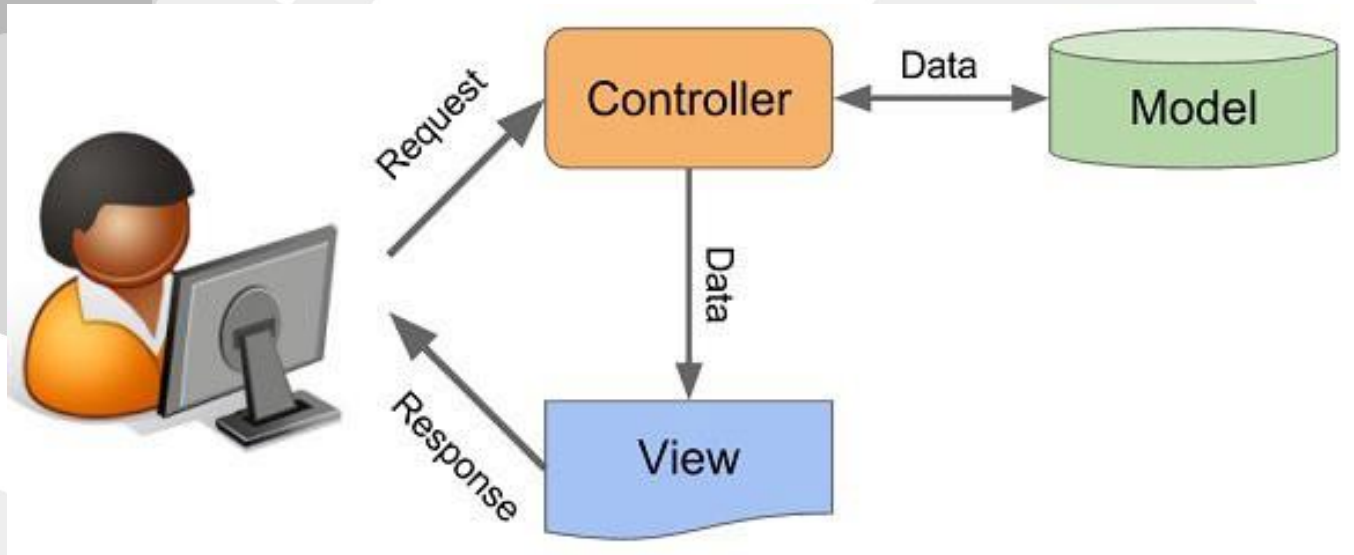
# MVC

Model, View, Controller

# MVC

- É um padrão de arquitetura de software que separa a representação da informação da interação do usuário.

# MVC





# MVC

- Observe a pasta **app** do seu projeto Rails

# **Criando o primeiro CRUD usando Scaffold**

# CRUD ?

**C**reate, **R**ead, **U**ppdate, **D**eleete  
Operações básicas em um BD/tabela

# Scaffold

O rails possui um ***generator*** chamado ***scaffold*** que permite criar um CRUD para uma determinada "tabela"

# Scaffold



# Scaffold

- Para usar o *generate* para *scaffold* do Rails, rode o comando...
  - `rails generate scaffold <Model> <campo:tipo>`  
`<campo:tipo> ...`

# Scaffold

- Transformando a modelagem em scaffold...

## Moedas

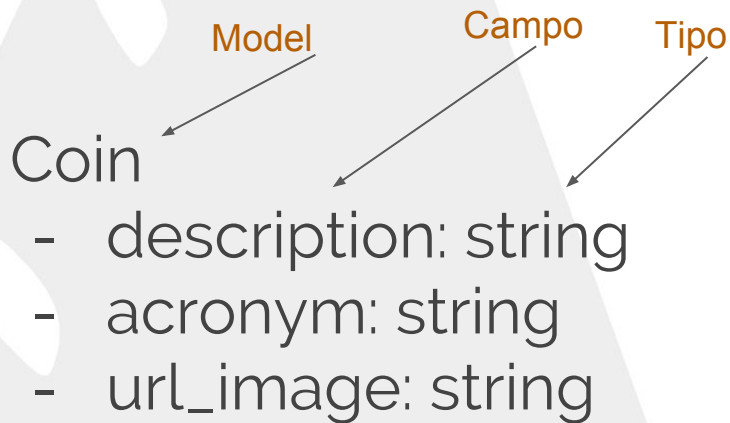
Nome	Sigla	Imagem
Bitcoin	BTC	<a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>
Litecoin	LTC	<a href="http://site.com/litecoin.jpg">http://site.com/litecoin.jpg</a>

# Scaffold

- Transformando a modelagem em scaffold...

## Moedas

Nome	Sigla	Imagem
Bitcoin	BTC	<a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>
Litecoin	LTC	<a href="http://site.com/litecoin.jpg">http://site.com/litecoin.jpg</a>





# Scaffold

- Programme em inglês
  - Dê preferência de escrever todos os models em inglês

# Scaffold

- Convenção
  - Um model é sempre escrito com a primeira letra maiúscula e no singular.
    - **Coin**

# Scaffold

- Tipos dos campos

- `http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html#method-i-add\_column`
- `description: string`
- `acronym: string`
- `url_image: string`

# Scaffold

- Sento assim...

- rails generate scaffold Coin description:string  
acronym:string url\_image:string

ou

- rails generate scaffold Coin description acronym  
url\_image

# Scaffold

- O comando anterior cria diversos arquivos que podem ser acompanhados pelo log, e dentre eles temos:
  - 1- As views
  - 2- O controller
  - 3- O model
  - 4- A **migração (migration)**

# **Entendendo sobre Migrations e ORM**

The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

# Migrations

# Migrations

- Quando falamos sobre Migrations precisamos falar sobre **Active Record** e **ORM**



The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

# **Active Record**

# Active Record

- Active Record é o **M** do MVC. O model.
- É a camada de software responsável pela lógica de dados e negócio.
- `http://guides.rubyonrails.org/v5.2/active\_record\_basics.html`

# Active Record

- Não confunda o **padrão** Active Record com o **framework** Active Record!
- Nesse caso o framework é a implementação do padrão!
- [https://pt.wikipedia.org/wiki/Active\\_record](https://pt.wikipedia.org/wiki/Active_record)

The background features a complex geometric design. On the left, there are several overlapping triangles in shades of gray and white. A large, light gray shape resembling a gear or a stylized letter 'C' dominates the right side of the image. The word 'ORM' is centered in a bold, dark red font.

**ORM**

# ORM

- ORM vem de **Object-Relational Mapping**
- Em resumo, é uma técnica que mapeia os dados em um BD para classes/objetos na programação
- [https://pt.wikipedia.org/wiki/Mapeamento\\_objeto-relacional](https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional)

# **Voltando às Migrations**

# Migrations

- Em resumo, migrations são uma característica do Active Record (o framework) que permite você "escrever/especificar" as tabelas do BD usando a linguagem Ruby.
- Dessa forma, você pode adicionar, modificar e remover tabelas do BD sem utilizar SQL, além de ter sempre disponível toda a sequência de criação/alteração das tabelas envolvidas no projeto.
- As migrações também controlam quais já foram ou não aplicadas o BD através do arquivo **db/schema.rb**.
- <http://guides.rubyonrails.org/v5.2/migrations.html>

# Migrations

- As migrations ficam localizadas em **db/migrate** em seu projeto. Confira.
- Sempre que criamos migrações, precisamos fazê-las "migrar" para o BD, ou seja, aplicar as mudanças no BD.
- Para isso, usamos algumas **tasks** predefinidas no Rails.



# **Rails dbconsole, Rails Tasks e primeiro acesso**

# Rails dbconsole

**rails dbconsole** é o comando usado se conectar ao BD e executar comandos para inspecioná-lo.

# Rails Tasks?

São tarefas predefinidas que o Rails pode executar

# Rails Tasks

- Para conhecer todas as tasks disponíveis rode
  - `rails -T`
- Para filtrar tasks específicas use as iniciais do que procura, por exemplo:
  - `rails -T db`

# Rails Tasks

- As tasks de banco de dados mais usadas são:
  - `rails db:create` #cria o db
  - `rails db:drop` #apaga o db
  - `rails db:migrate` #executa as migrations
  - `rails db:rollback` #desfaz a última migration

# Rails Tasks

- Sendo assim...
  - Precisamos criar o BD a primeira vez.
    - `rails db:create`
  - Na sequência aplicamos as migrações.
    - `rails db:migrate`

# Inicie o servidor do Rails

`rails server`

# Acesse a rota *coins*

`http://localhost:3000/coins`





# Ambientes do Rails



# Ambientes do Rails

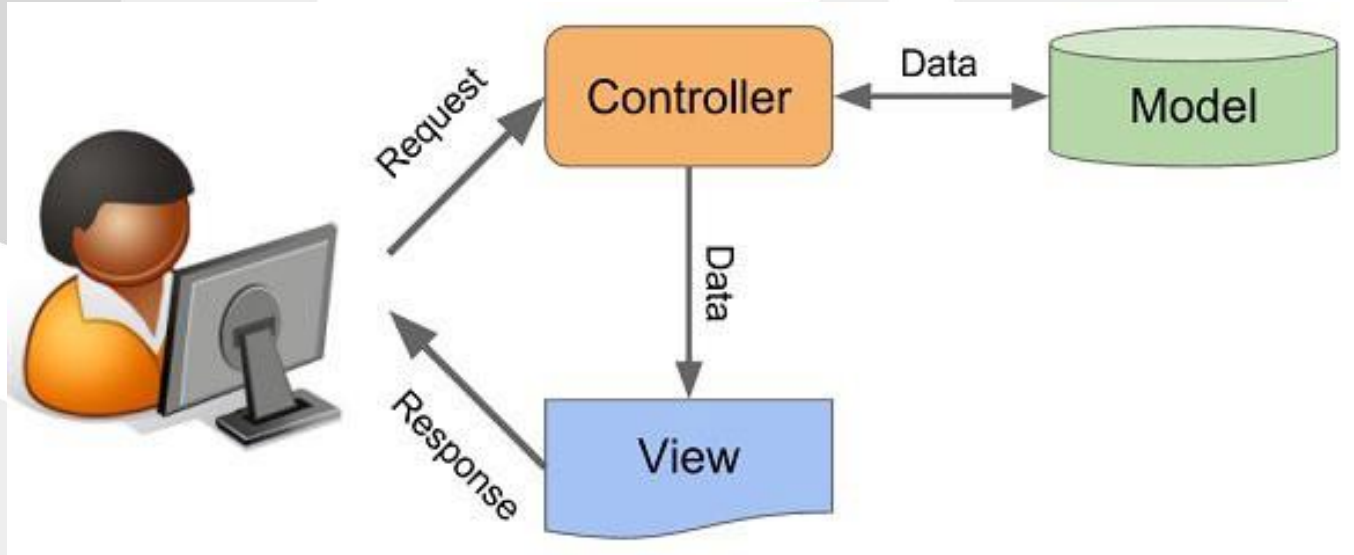
# Ambientes do Rails

- Por padrão, o Rails vem com 3 ambientes: **Desenvolvimento, Teste e Produção.**
- Esses ambientes possibilitam ter configurações isoladas para cada ambiente.
- A sua Gemfile é a forma mais simples de perceber isso.
- `http://guides.rubyonrails.org/configuring.html#creating-rails-environments`

# **Fluxo MVC e "acesso padrão" aos controllers**

# Fluxo MVC

- Lembra do padrão MVC?



## Fluxo MVC

- Após criarmos nosso *scaffold*, ganhamos model, views e controller para "Coin" que foi o nome escolhido para o nosso model. Observe no projeto.
- Daí, usamos o path **/coins** para acessar a página principal.
- Isso ocorre por alguns motivos, dentre eles é que o controller foi gerado com o nome **coins\_controller** (a convenção do Rails é ter controllers sempre arquivos com **???\_controller.rb**)

# Fluxo MVC

- Dentro do controller temos algumas actions (que são métodos) e dentre elas uma chamada ***index*** (que é a action "padrão")
- Sendo assim, ao acessar **/coins** pelo navegador indica que vamos para o controller coins e por ser um acesso "padrão" a action index será invocada, disparando na sequência o arquivo da view com o mesmo nome, nesse caso **app/views/coins/index.html.erb**

# Fluxo MVC

- Entender esse fluxo inicial é de suma importância para o entendimento do Rails como um todo!



# **Novos generators e tela de Boas Vindas**



# **Conhecendo novos generators**

# Conhecendo novos generators

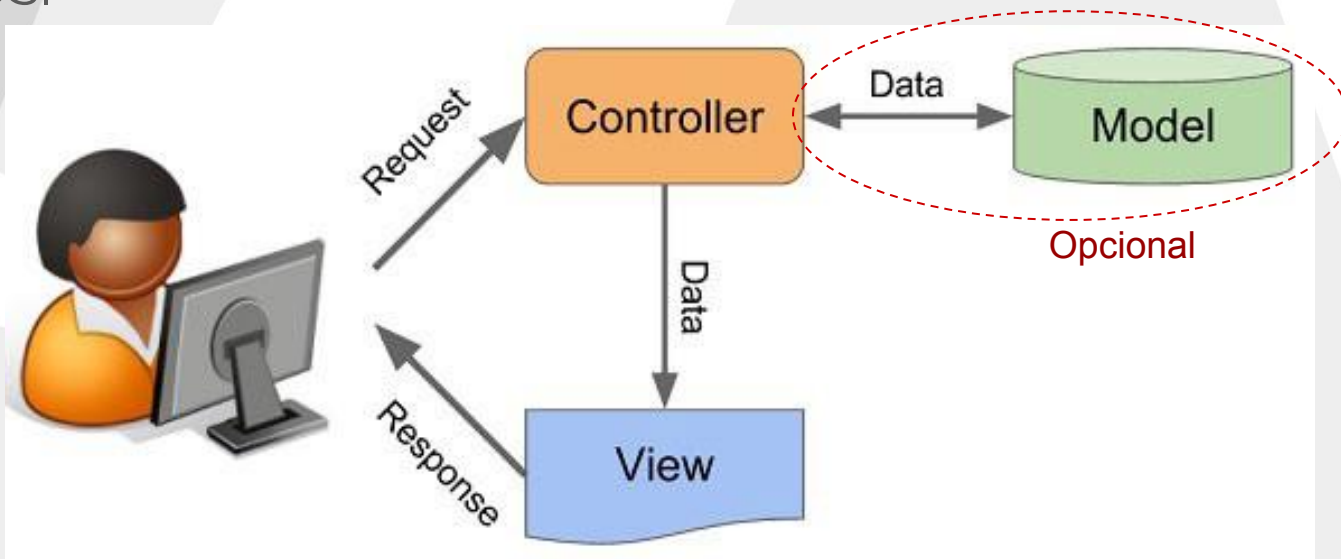
- No rails, podemos conhecer novos generators simplesmente digitando um dos comandos abaixo...
  - `rails generate`
  - `rails g`
- Ao digitar o comando acima, o rails mostrará os generators disponíveis.
- Novos generators podem ser adicionados ao usarmos algumas gems (geralmente a documentação da gem indica isso).

## Conhecendo novos generators

- Dentre os generators apresentados, temos o generator do scaffold (MVC), do model e do controller.
- Mas cadê o generator das views?
- Vamos entender isso no próximo slide, mas antes, teste criar um controller e depois destruí-lo usando o `rails destroy`

# Conhecendo novos generators

- Perceba que a conversação do Controller com o Model é opcional mas o Controller e a View sempre andam juntos.





# **Gerando a tela de Boas Vindas**

## Gerando a tela de Boas Vindas

- Agora que conhecemos como o Fluxo MVC funciona na aplicação Rails, sabemos que para cada action deve existir uma view associada, correto? Verifique no projeto.

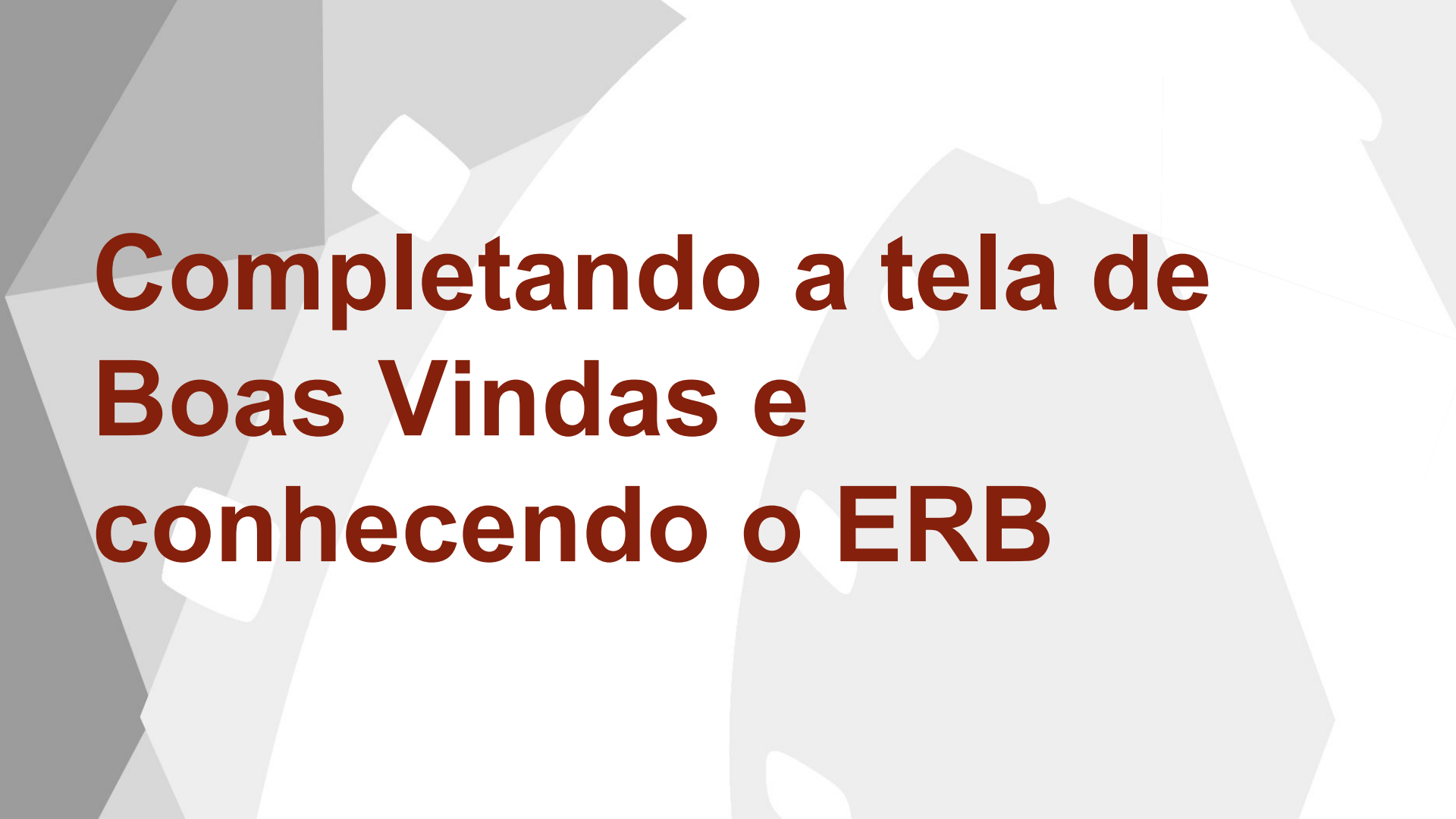
## Gerando a tela de Boas Vindas

- Em seguida, usando o generator do controller, faça:
  - `rails g controller welcome index`
- Usando dessa forma, indicamos que queremos um controller chamado welcome com uma action chamada index, e por consequência também será gerada a estrutura da view baseada na action.
- Simples, não? :-)



## Gerando a tela de Boas Vindas

- Agora basta adicionar o conteúdo na view, e pronto!  
Pode testar sua app novamente acessando **/welcome**
- Oops... não vai funcionar...
- Podemos então perceber na tela de erro que existe agora uma rota chamada **/welcome/index** que será a nossa rota correta.



**Completando a tela de  
Boas Vindas e  
conhecendo o ERB**



# ERB

Embedded Ruby

# ERB

- ERB vem de Embedded Ruby
- Em resumo, é uma forma de você mesclar texto com código Ruby.
- A princípio você pode achar que uma simples "interpolação" com `#{ }` resolveria, mas o ERB te dá a oportunidade de trabalhar com textos, HTML e expressões Ruby, ou seja, ele é muito mais completo e complexo do que uma simples interpolação.

# ERB

- Sendo assim, diz-se que ERB é um sistema de templates para o Ruby.
  - `https://pt.wikipedia.org/wiki/ERuby`
- Veja essa documentação:
  - `http://ruby-doc.org/stdlib-2.4.0/libdoc/erb/rdoc/ERB.html`

# ERB

- No nosso caso, o melhor é testar isso em nossa página de Boas Vindas.
- No arquivo **app/views/welcome/index.html.erb** digite:
  - `<%= 1 + 1 %>`
  - `<%= Date.today %>`
- Perceba que o formato do arquivo é HTML.ERB, ou seja serão usados HTML e Ruby no arquivo

# Helpers



# Helpers?

`http://api.rubyonrails.org/classes/ActionController/Helpers.html`



## O Helpers

- Em resumo, helpers são métodos prontos que podem ser usados nas views.
- Esse métodos geralmente facilitam a vida do programador, fazendo com que menos código seja escrito para que se consiga o mesmo resultado.
- Vamos conhecer o helper **link\_to**

# O Helpers

- **link\_to**

- [http://api.rubyonrails.org/v5.2/classes/ActionView/Helpers/UrlHelper.html#method-i-link\\_to](http://api.rubyonrails.org/v5.2/classes/ActionView/Helpers/UrlHelper.html#method-i-link_to)

```
<%= link_to "Cadastro de Moedas", coin_path %>
```



# **Um pouco mais sobre Helpers**

The background features a stylized, light gray coin with a ruler-like scale around its perimeter. The scale has several white rectangular markers. The coin is positioned diagonally, with its top-left corner towards the upper left of the frame.

**Vamos ajustar as imagens  
das moedas?**

# Ajustando as imagens

- Sabe-se que para colocar uma imagem em HTML usa-se por exemplo o...
  - ``
- Você pode conseguir o resultado esperado para as moedas, usando...
  - ``

# Ajustando as imagens

- No entanto, vamos usar o helper `image_tag` para conseguir o mesmo resultado...
  - `<%= image_tag coin.url_image %>`

# image\_tag

[https://api.rubyonrails.org/classes/ActionView/Helpers/AssetTagHelper.html#method-i-image\\_tag](https://api.rubyonrails.org/classes/ActionView/Helpers/AssetTagHelper.html#method-i-image_tag)



# **Criando seus próprios Helpers**



# Criando seus próprios Helpers

- O Rails permite que você crie seus próprios Helpers.
- Para isso, existe uma pasta **app/helpers** que permite esse feito.
- Perceba que essa pasta possui um arquivo **application\_helper.rb** e outros arquivos com o nome das views \*\_helper.rb

# Criando seus próprios Helpers

- Os nomes dos arquivos são para facilitar a organização na hora de criar seus helpers.
- Todos os helpers criados serão disponibilizados para as views.

The background features a light gray abstract design with overlapping geometric shapes. A faint, stylized outline of a globe is visible, showing continents and latitude/longitude lines. Several white, rounded rectangular shapes are scattered across the composition, some appearing as if they are floating or attached to the geometric forms.

**Crie seu próprio Helper**

# Crie seu próprio Helper

- Dentro do arquivo ***application\_helper.rb*** crie uma função conforme abaixo...
  - `def data_br(data_us)`
  - `data_us.strftime("%d/%m/%Y")`
  - `end`
- **Documentação strftime**
  - <https://apidock.com/ruby/DateTime/strftime>

# **Rotas e REST**

# HTTP

Protocolo padrão de acesso a Internet

# HTTP - Verbos

GET / POST

# Verbos

- **GET:** Usado quando digitamos diretamente a URL no navegador
- **POST:** Usado por formulários da Web (dados enviados por 'debaixo dos panos')



# Verbos

- E para apagar? **GET** ou **POST**?

The background features a complex arrangement of overlapping geometric shapes in various shades of gray, from light to dark. These shapes include triangles, polygons, and curved forms that create a sense of depth and movement. The word "REST" is centered in a bold, dark red font, providing a strong contrast against the lighter background elements.

**REST**

# REST

- Representational State Transfer
- Transferência de Estado Representacional
- 2000 - Roy Fielding
- Idealizou que o HTTP deveria ter 1 verbo para cada uma das ações do CRUD

# REST



CREATE

C



READ

R



UPDATE

U



DELETE

D

# REST

HTTP/ REST	CRUD	SQL
POST	Create	INSERT
GET	Retrieve	SELECT
PUT	Update	UPDATE
DELETE	Delete	DELETE

The background features a complex arrangement of overlapping geometric shapes in various shades of gray. A large, light-gray shape resembling a gear or a stylized letter 'C' is prominent on the right side. Several smaller, darker gray shapes are scattered on the left and top. The word "Recurso" is centered in a bold, dark red font.

**Recurso**

# Recurso

## Moeda

HTTP/ REST	CRUD	SQL
POST	Create	INSERT
GET	Retrieve	SELECT
PUT	Update	UPDATE
DELETE	Delete	DELETE

# Recurso

- Veja o mapeamento das rotas do Rails
  - `http://guides.rubyonrails.org/routing.html#crud-verbs-and-actions`



# Recurso

`http://localhost:3000/rails/info/routes`



# Rotas

# Rotas

- Vamos começar fazendo o mapeamento da rota padrão.
- O arquivo do Rails que faz o mapeamento das rotas é o **config/routes.rb**

# Rotas

- Para fazermos o mapeamento padrão devemos usar a instrução **root to:**
  - `root to: "welcome#index"`
- Documentação:
  - `http://guides.rubyonrails.org/routing.html#using-root`

# Aprenda mais sobre REST!

<https://www.videosdeti.com.br/restful-apis.html>

The background features a light gray map of Brazil, which is partially obscured by several overlapping, semi-transparent geometric shapes in shades of gray. These shapes include triangles and polygons of various sizes, creating a layered, abstract effect. The text is centered over the map.

# **Um pouco mais sobre Rotas**

# **Resources no Rails**

[https://guides.rubyonrails.org/routing.html#cr  
ud-verbs-and-actions](https://guides.rubyonrails.org/routing.html#crud-verbs-and-actions)

## Resources

- Quando usamos a palavra **resources** no mapeamento das rotas do Rails, de forma automática ele entenderá que queremos criar 7 rotas padrão para o CRUD.



The background features a light gray abstract design with overlapping geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the upper right corner. The text is centered in a bold, dark red font.

**Crie suas próprias Rotas**

# Crie suas próprias rotas

- Para criar suas próprias rotas apenas declare o verbo e a url e o controller e action que deve responder à url indicada, veja o exemplo:
  - `get '/inicio', to: 'welcome#index'`
- A rota acima permitirá que seja digitada a url `/inicio` e ela acessará o controller **welcome** e ação **index**.



# **Rails Console**

# Rails Console?

[https://guides.rubyonrails.org/command\\_line.html#rails-console](https://guides.rubyonrails.org/command_line.html#rails-console)

# Rails Console

- O Rails Console permite você interagir com sua aplicação a partir do terminal.
  - `rails console`
  - `rails c`

# Rails Console

- Você pode rodar o rails console com a opção **-e** passando o ambiente que quer usar
  - rails console -e production

# Rails Console

- Você pode também usar o rails console com a opção **--sandbox** para que nenhum dado seja alterado na aplicação
  - rails console --sandbox

# Rails Console

- O Rails console também permite você usar os objetos `app` e `helper`, a fim de testar por exemplo os paths e verificar as saídas dos helpers depois de processado.

Veja:

- `app.root_path`
- `helper.link_to "teste", "/teste"`
- `helper.data_br(Date.today)`





# **Trabalhando com Models via Rails Console**

# Trabalhando com Models via Rails Console

- No Rails console podemos trabalhar com qualquer model, bastando para isso invocar o seu nome e usar os métodos disponíveis a partir do Active Record.
  - `Coin.first` # retorna o primeiro elemento
  - `Coin.last` # retorna o último elemento
  - `Coin.all` # retorna todos os elementos

# Trabalhando com Models via Rails Console

- Perceba que os métodos anteriores `first` e `last` retornam um único elemento, já o método `all` retorna um **array** de elementos.
  - `Coin.all`     # retorna todos os elementos
- Ou seja, se for necessário, itere no array para conseguir cada um dos elementos.
  - `Coin.all.each do |coin|`
  - `puts coin`
  - `end`

# Trabalhando com Models via Rails Console

- Veja mais alguns usos.
- Criando uma nova moeda.
  - `c = Coin.new`
  - `c.description = "Dash"`
  - `c.acronym = "DASH"`
  - `c.url_image = "http://"`
  - `c.save!`

# Trabalhando com Models via Rails Console

- Outra forma de criar uma nova moeda.
  - `c = Coin.create!`
  - `description: "Dash",`
  - `acronym: "DASH",`
  - `url_image: "http://"`
  - `)`

# Trabalhando com Models via Rails Console

- Você pode explorar outros métodos digitando nome do model com um ponto e pressionando TAB.
  - `Coin.`
  - `Display all 616 possibilities? (y or n)`

# Identificando os "ambientes" no Rails



**Como identificar?**



## Como identificar?

- Quando estamos trabalhando no Rails, é interessante que possamos detectar em qual ambiente do Rails estamos, visto que muitas vezes podemos tomar algumas decisões, como por exemplo, mostrar ou não um menu, calcular algo diferente quando estiver em produção e assim vai...

## Como identificar?

- Para resolver esse problema podemos simplesmente usar a instrução...
  - `Rails.env`
- Ou ainda...
  - `Rails.env.production?`
  - `Rails.env.development?`

<https://api.rubyonrails.org/v5.2.0/classes/Rails.html#method-c-env>

# **Variáveis de instância/sessão no Rails**

# Variáveis de instância/sessão no Rails

- As variáveis que começam com "@" são variáveis de instância quando usamos a Orientação a Objetos, lembram?
- ```
class Pessoa
  o @nome = ""
  o def initialize(nome)
  o   @nome = nome
  o end
  o def meu_nome
  o   @nome
  o end
end
```

## Variáveis de instância/sessão no Rails

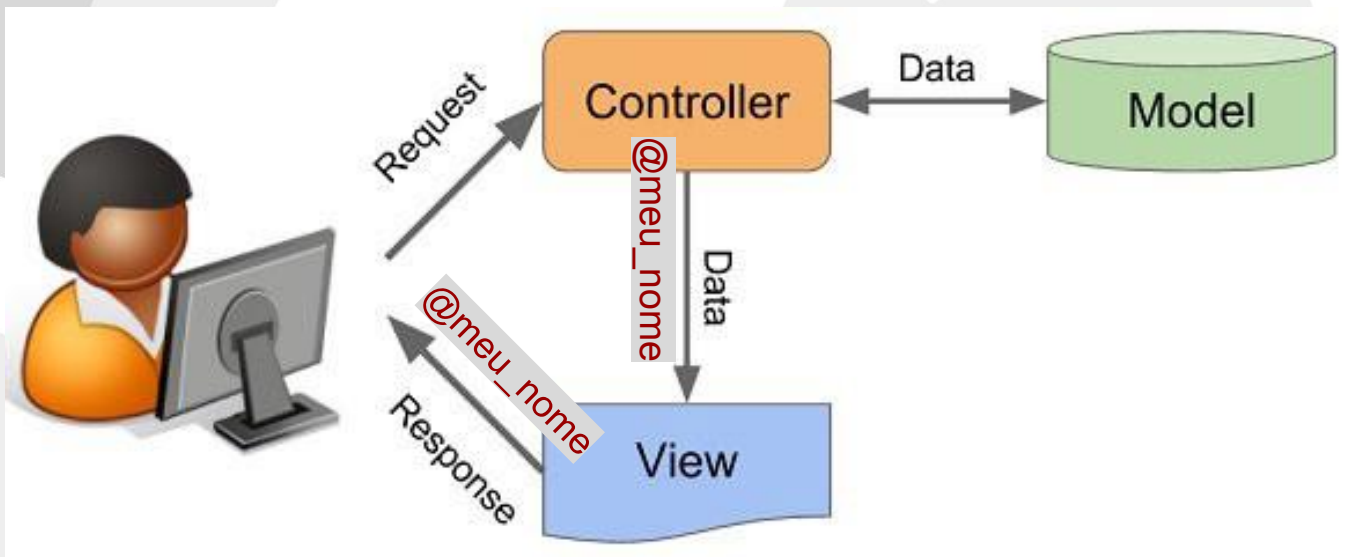
- Pois bem, no caso do Rails é comum chamarmos as variáveis que começam com "@" de variáveis de sessão, visto que essas variáveis terão um valor único para cada sessão do navegador que for aberta.

## Variáveis de instância/sessão no Rails

- Observe o **app/controllers/coins\_controller.rb**
- Para fazer um teste, nesse mesmo arquivo, no método **index** crie uma variável **@meu\_nome = "Jackson Pires"**
- Agora na view **app/views/coins/index.html.erb** adicione a linha **<%= @meu\_nome %>** e salve.
- Perceba que ao acessar a página, agora seu nome aparecerá.

# Variáveis de instância/sessão no Rails

- Podemos dizer, por exemplo, que essas variáveis vão servir para o **response** da nossa requisição.





# Query Params para Requests



The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

**Lembram dos Hashes?**

# Hashes

- `h = {nome: "Jackson", curso: "Rails"}`
- `h[:nome]`
- `h[:curso]`

# Query Params

## Query Params para Requests

- Query Params são parâmetros enviados para o servidor através da URL.
- Para isso basta usar, logo após a URL padrão, o símbolo de interrogação "?" seguido de um par **chave=valor** para o que se quer passar para o servidor.

# Query Params para Requests

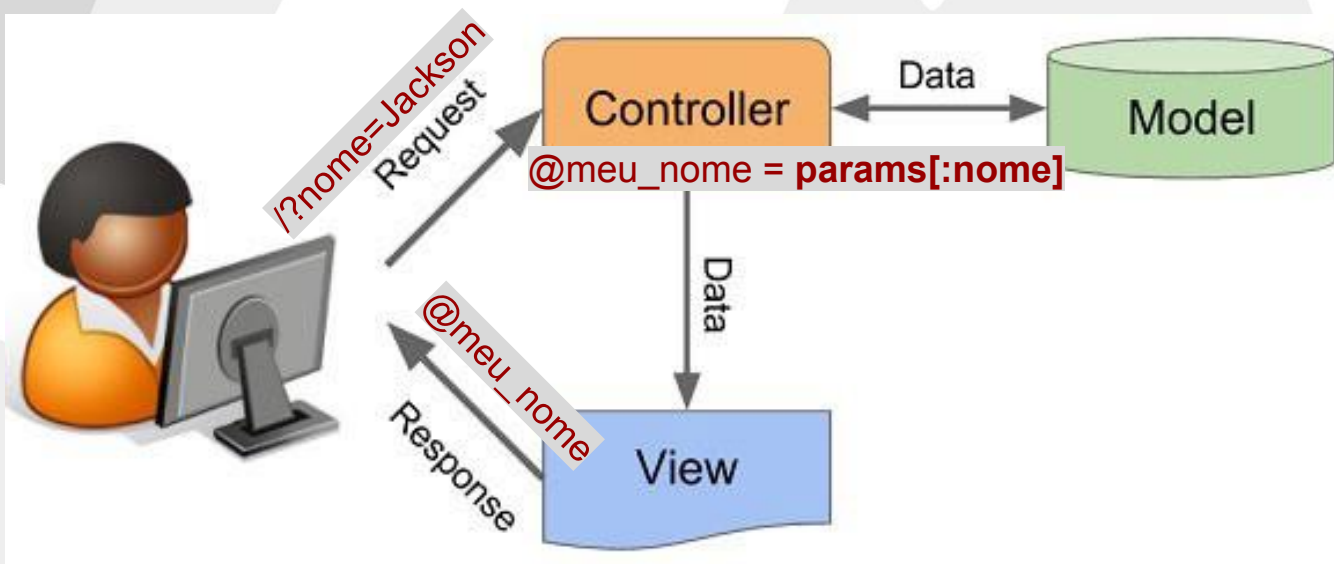
- Vejamos o exemplo...
- `http://localhost:3000/?nome=Jackson`
- Acessando a URL acima, podemos informar o parâmetro nome para o servidor.
- No servidor, no controller, podemos acessá-lo através da variável global params.
  - `params[:nome]`
- Assim podemos usar o valor que foi passado (**Jackson**) do jeito que quisermos.

# Query Params para Requests

- Uma pequena documentação pode ser obtida aqui:
- `https://guides.rubyonrails.org/action\_controller\_overview.html#parameters`

# Query Params para Requests

- Sendo assim, entendemos query params dessa forma:



# Partials





# Partials

- Partials permitem você renderizar uma view dentro de outra, ou seja, "reutilizar páginas/views"
- Para isso crie um arquivo **\_menu.html.erb**
- Perceba o \_ "underline" no início do nome do arquivo
- Dentro dele coloque o conteúdo que está entre **<ul>**  
**</ul>** da página **app/views/welcome/index.html.erb**

# Partials

- Agora na página ... use o helper **render** para indicar o nome da partial para carregar o menu.
- `<%= render "menu" %>`

# Partials

- Documentação
- [https://guides.rubyonrails.org/layouts\\_and\\_rendering.html#using-partials](https://guides.rubyonrails.org/layouts_and_rendering.html#using-partials)

# Layouts

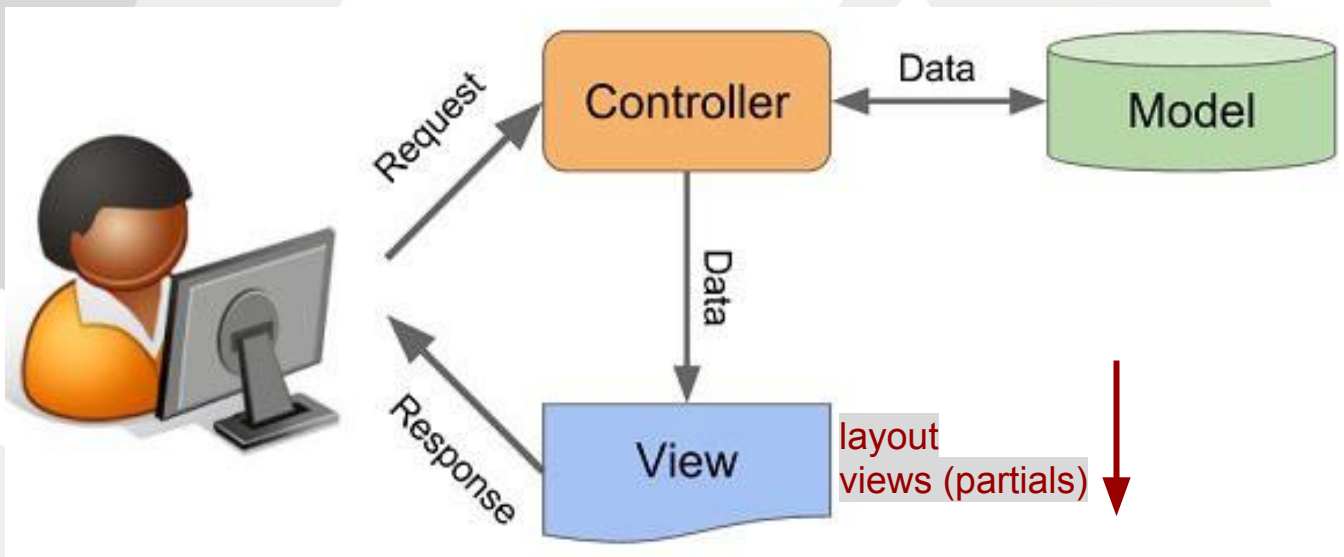


# Layouts

- Quando acessamos uma URL o Rails, no momento em que vai enviar a resposta para o usuário, renderiza um layout padrão (application) antes de renderizar a view que solicitamos.

# Layouts

- Vejamos:



# Layouts

- Se quisermos alterar o layout padrão, basta no **controller** indicarmos o layout que deseja-se renderizar.
  - `layout "nome_do_layout"`

# Layouts

- Documentação
  - <https://api.rubyonrails.org/classes/ActionView/Layouts.html>
  - [https://guides.rubyonrails.org/layouts\\_and\\_rendering.html](https://guides.rubyonrails.org/layouts_and_rendering.html)





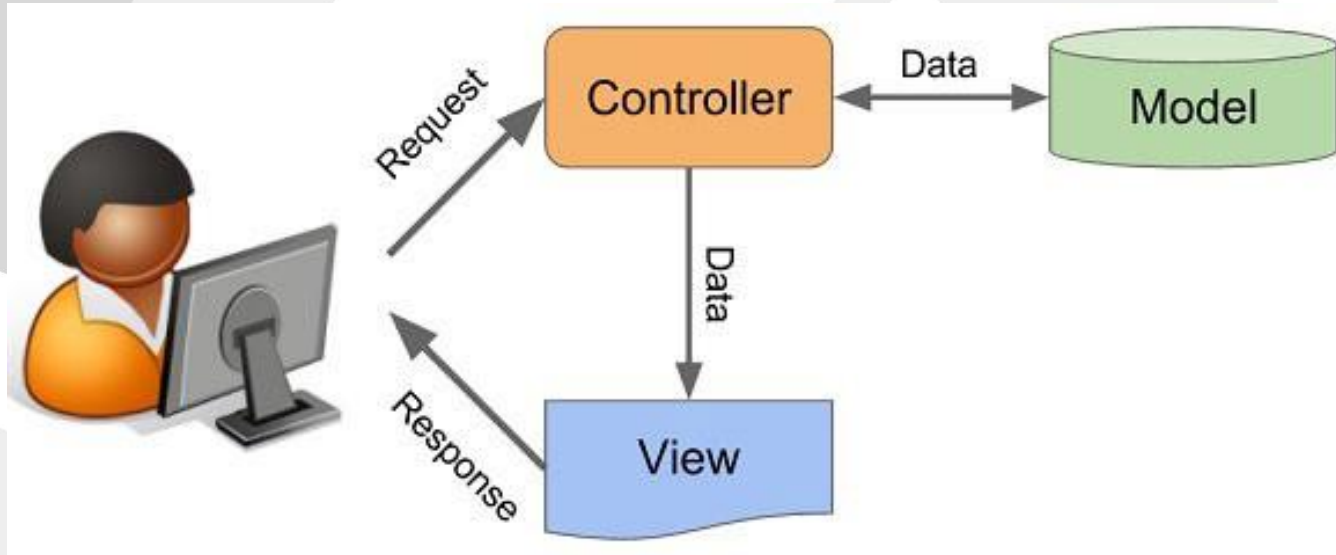
# **Fluxo MVC do CRUD (Index)**

## Fluxo MVC do CRUD (Index)

- É importante entender o fluxo MVC que o Rails usa, sendo assim, vamos voltar agora a entender esse fluxo.
- Nesse momento em que já estamos mais familiarizados com o Rails tudo fará mais sentido ao analisarmos o CRUD gerado pelo Scaffold.
- Vejamos...

# Fluxo MVC do CRUD (Index)

- Vejamos...



## Fluxo MVC do CRUD (Index)

- Tudo começa com uma requisição (**request**) que o usuário faz.
- Essa requisição é **roteada para um controller** e uma action.
- A action/controller é processada e o resultado (**response**) é enviado para o usuário.

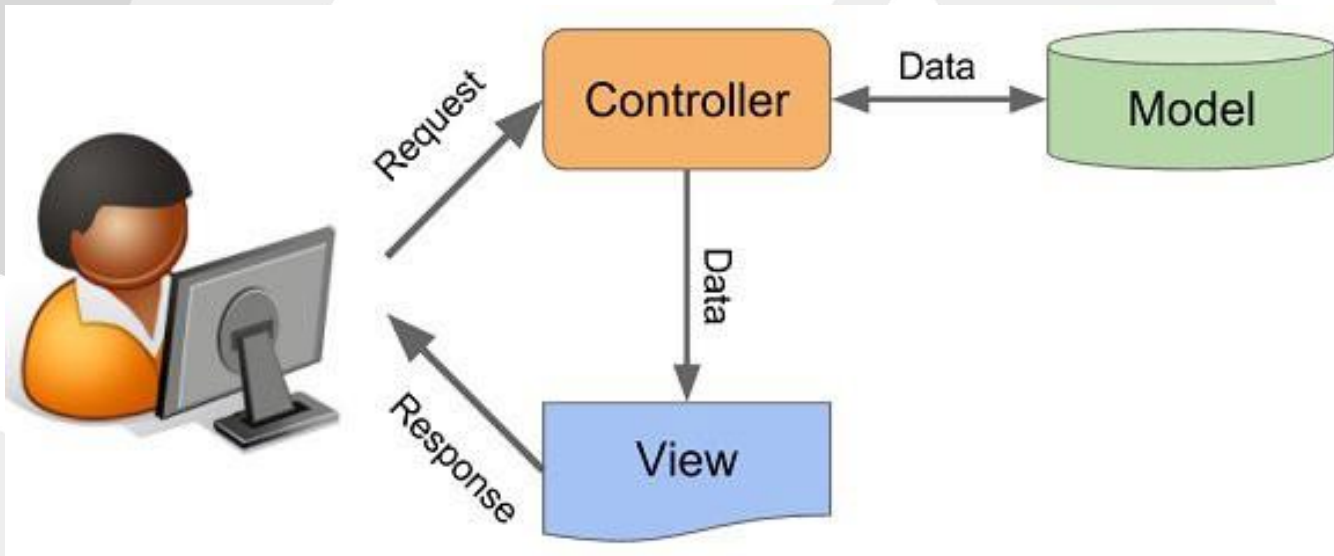
## Fluxo MVC do CRUD (Index)

- No caso da action **index**, todas as moedas são buscadas no model **Coin** e enviadas através da variável de instância **@coins** .
- Ao chegar na view, a variável **@coins** é iterada e **processada junto ao HTML** para que sejam mostradas todas as moedas.

# **Fluxo MVC do CRUD (Show e Delete) + Filtros Rails**

# Fluxo MVC do CRUD

- Vejamos...



## Fluxo MVC do CRUD (Show)

- No caso da action **show**, a request é feita em conjunto com um ID que identifica qual a moeda que será mostrada.
- Ao chegar no controller a moeda é pesquisada pelo ID e enviada para a view através da variável **@coin**
- O Rails usa o conceito de filtros para buscar a moeda
  - [https://guides.rubyonrails.org/action\\_controller\\_overview.html#filters](https://guides.rubyonrails.org/action_controller_overview.html#filters)
- Ao chegar na view, os dados da moeda são mesclados ao HTML



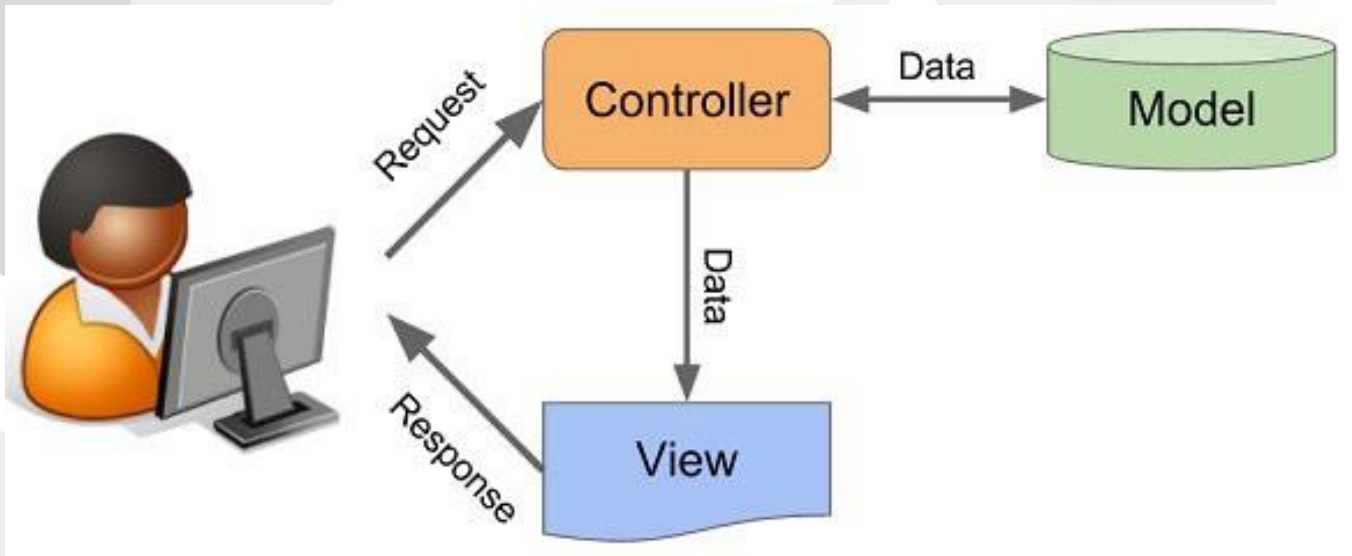
## Fluxo MVC do CRUD (Delete)

- No caso da action **delete**, a request também é feita em conjunto com um ID que identifica a moeda.
- Ao chegar no controller a moeda é pesquisada pelo ID e apagada e a requisição é redirecionada para a index novamente.

# **Fluxo MVC do CRUD (New e Create)**

# Fluxo MVC do CRUD

- Vejamos...



## Fluxo MVC do CRUD (New)

- Para gerar uma nova moeda será usado inicialmente um path para um novo elemento **coins/new**.
- Ao acessar esse path, o controller **gera uma instância vazia do model Coin (Coin.new)**.
- Isso fará com que a variável **@coin** possua todos os campos (vazios) e permitirá que o Rails possa montar a view (**new.html.erb**) com um formulário para a nova moeda.

## Fluxo MVC do CRUD (Create)

- O formulário é montado com o helper **form\_with** (novo no Rails 5)
  - `https://api.rubyonrails.org/v5.1/classes/ActionView/Helpers/FormHelper.html#method-i-form\_with`
- Após preencher o formulário e clicar no botão para cadastrar a requisição do tipo **POST** será enviada ao controller que rodará a action create.

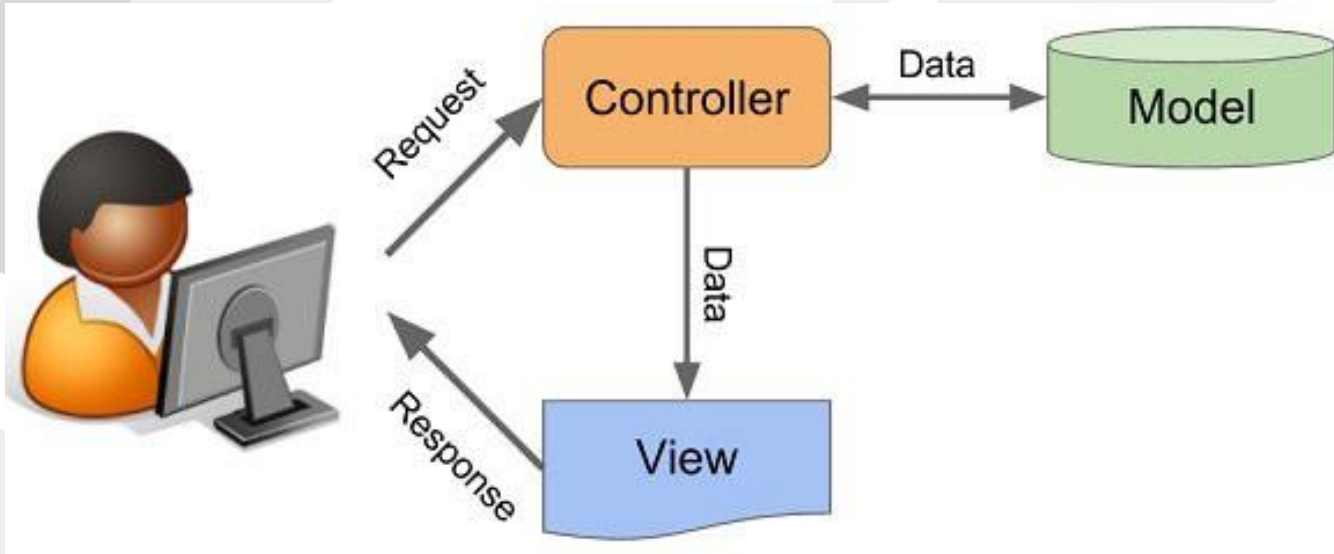
## Fluxo MVC do CRUD (Create)

- Lá a nova moeda é criada com os dados que foram enviados do formulário. Tudo através da variável **params**.
- Ao final a requisição é redirecionada para o **path show**, a fim de mostrar a moeda criada.

# **Fluxo MVC do CRUD (Edit e Update)**

# Fluxo MVC do CRUD

- Vejamos...





## Fluxo MVC do CRUD (Edit e Update)

- Tudo começa com uma requisição (**request**) que o usuário faz para editar uma moeda.
- A rota nos leva até a action **edit** que encontra o usuário e envia através da variável **@coin** a moeda a ser editada para a view **edit.html.erb**.

## Fluxo MVC do CRUD (Edit e Update)

- Após editar a moeda na view o usuário clicará no botão para atualizá-la.
- O botão submeterá as alterações via **PATCH** para a action **update**, que localiza e atualiza a moeda com os dados recebidos via **params**.
- Após atualizar, o usuário é redirecionado para o **show**, que mostra os dados atuais da moeda.

# **Um pouco mais sobre "Permissão de parâmetros"**

# Um pouco mais sobre "Permissão de parâmetros"

- A permissão de parâmetros é uma prática que visa informar quais dados que estão chegando ao controller são permitidos para serem manipulados.
- O Rails trata isso através do framework ActionController e nos dá os métodos **require** e **permit** para usarmos com essa finalidade.
- <https://api.rubyonrails.org/classes/ActionController/Parameters.html>

# Um pouco mais sobre "Permissão de parâmetros"

- Veja o Exemplo:

- `params = ActionController::Parameters.new({`
- `person: {`
- `name: "Francesco",`
- `age: 22,`
- `role: "admin"`
- `}`
- `})`
- 
- `permitted = params.require(:person).permit(:name, :age)`
- `permitted`

The background features a complex arrangement of overlapping geometric shapes in various shades of gray, creating a sense of depth and movement. A white film strip with several rectangular frames is draped across the composition, adding a cinematic or archival feel. The overall aesthetic is modern and minimalist.

**A task db:seed**

# rails db:seed

`https://guides.rubyonrails.org/active\_record\_migrations.html  
#migrations-and-seed-data`

## A task db:seed

- Imagine a seguinte situação... Você começa a testar seu software cadastrando novas moedas com todos os dados, etc.
- Em dado momento, você precisa apagar todos os dados para corrigir algo no BD, ou mesmo, passa seu projeto para alguém sem os dados.



## A task db:seed

- O esforço e tempo dedicados para se cadastrar tudo novamente pode ser muito grande, visto que na medida em que o software cresce, também cresce a complexidade dos dados envolvidos, como por exemplo, primeiro deve-se cadastrar as moedas, depois os usuários, depois os dependentes dos usuários e assim vai.

## A task db:seed

- Além disso, uma aplicação normalmente já vem com "dados iniciais", como por exemplo o nome e sigla dos estados (caso haja) ou qualquer outro dado que faça sentido no sistema já vem por padrão.
- A fim de solucionar esse problema podemos fazer a alimentação inicial do BD sempre que a aplicação for iniciada a primeira vez.

## A task db:seed

- O Rails já possui um mecanismo para isso chamado "**db:seed**" que é a task que podemos invocar para fazer o preenchimento inicial dos dados.

## A task db:seed

- Para o nosso caso, vamos criar mecanismo para que os caso apaguemos todos os dados, possamos preencher com novos sem muito esforço.
- Sendo assim, vá em **db/seed.rb** e adicione as seguintes linhas:

## A task db:seed

- `Coin.create!(`
  - `description: "Bitcoin"`
  - `acronym: "BTC"`
  - `url_image:`  
`"https://upload.wikimedia.org/wikipedia/commons/c/cf/Bitcoin.com_logo.png"`
- `)`

# A task db:seed

- Para fazer um teste, rode:
  - `rails db:drop db:create db:migrate db:seed`



# **Criando suas próprias Rake Tasks**



**Antes...**



## O %x

- O %x (minúsculo) permite que executemos comando no terminal a partir de um código ruby.
- Para testar, abra o IRB e digite a instrução:
  - `puts %x(ls)`
- O resultado deve ser a lista de pastas da sua aplicação

# Criando suas próprias Rake Tasks

[https://guides.rubyonrails.org/command\\_line.  
html#custom-rake-tasks](https://guides.rubyonrails.org/command_line.html#custom-rake-tasks)

# Criando suas próprias Rake Tasks

- No Rails é possível criar suas próprias tasks de forma muito simples. Para isso rode:
  - `rails g task <nome do namespace> <nome da task>`
  - Ex:
    - `rails g task dev setup`
  - Isso gerará a task **dev:setup** que pode ser usada para criarmos tudo necessário para a configuração do ambiente de desenvolvimento

# Criando suas próprias Rake Tasks

- Agora use o %x para invocar comandos do terminal
  - `if Rails.env.development?`
    - `%x(rake db:drop db:create db:migrate db:seed)`
  - `end`
- Ao final faça um teste
  - `rails dev:setup`



# **Melhorando nossa Rake Task com as TTY Gems**

# TTY Gems

<https://piotrmurach.github.io/tty/>

# TTY Gems

- É um conjunto de gems focadas em terminal / CLI
- Vamos usar a gem **tty-spinner**
  - <https://github.com/piotrmurach/tty-spinner>
- Veja esse post onde falo um pouco sobre essa gem
  - <http://bit.ly/rake-tasks-elegantes>

# TTY Gems

- Vamos aproveitar para melhorar a nossa task **dev:setup** adicionando o código abaixo
  - `spinner = TTY::Spinner.new("[:spinner] Cadastrando moedas...")`
  - `spinner.auto_spin`
  - ***<código de criação das moedas>***
  - `spinner.success(' (Concluído!) ')`





# **Refatorando nossa Rake Task**

# Refatorando nossa Rake Task

- Ao finalizar a aula anterior, percebemos o código recebeu muitas duplicações e isso não é legal.
- Sendo assim, vamos melhorar o nosso código.
- Primeiro, crie um método no arquivo dev.rake
  - `def show_spinner(msg_start, msg_end = "Concluído")`
  - `spinner = TTY::Spinner.new("[:spinner]`
  - `#{msg_start}")`
  - `yield`
  - `spinner.auto_spin`
  - `spinner.success(msg_end)`
  - `end`

# Refatorando nossa Rake Task

- Depois basta chamar o método quando precisar...
  - `show_spinner("Apagando BD...") do`
    - `%x(rails db:drop)`
  - `end`

# **Refatorando o seeds.rb (find\_or\_create\_by)**

## Refatorando o seeds.rb (find\_or\_create\_by)

- Chegou a hora de melhorarmos também o **seeds.rb**
- Para isso podemos antes de mais nada diminuir a quantidade de duplicações usando o create com um Array de Hashes, assim, por exemplo:

```
Coin.create!([{description: "Bitcoin"},  
{description: "Ethereum"}])
```

## Refatorando o seeds.rb (find\_or\_create\_by)

- Mas, ainda assim se você rodar o rails db:seeds vai perceber um importante problema.
- Sempre que rodamos o referido comando as moedas são cadastradas repetidas em nosso sistema pois não há uma verificação que avalie se já existem moedas cadastradas.
- Para nossa sorte isso é bem fácil de resolver.

## Refatorando o seeds.rb (find\_or\_create\_by)

- O Active Record nos provê um método chamado `find_or_create_by` que permite pesquisar se um determinado registro já existe antes de cadastrá-lo.
- [https://api.rubyonrails.org/classes/ActiveRecord/Relation.html#method-i-find\\_or\\_create\\_by](https://api.rubyonrails.org/classes/ActiveRecord/Relation.html#method-i-find_or_create_by)
- [https://guides.rubyonrails.org/active\\_record\\_querying.html#find-or-create-by](https://guides.rubyonrails.org/active_record_querying.html#find-or-create-by)

## Refatorando o seeds.rb (find\_or\_create\_by)

- Faça o teste com uma moeda...
- `Coin.find_or_create_by!`
  - `description: "Bitcoin",`
  - `acronym: "BTC",`
  - `url_image:`  
`"https://assets.chinatechnews.com/wp-content/uploads/bitc`  
`oin-logo.jpg"`
- `)`



## Refatorando o seeds.rb (find\_or\_create\_by)

- Aqui percebemos um pequeno problema.
- Como vamos utilizar o nosso array de hashes nesse caso?
- Simples, utilize o each e rode o find\_or\_create\_by! em cada um dos elementos do array.

## Refatorando o seeds.rb (find\_or\_create\_by)

- `coins = [{description: "Bitcoin"}, {description: "Ethereum"}] #exemplo`
- `coins.each do |coin|`
  - `Coin.find_or_create_by!(coin)`
- `end`

## Refatorando o seeds.rb (find\_or\_create\_by)

- Por fim, altere o seeds para usar a gem **tty-spinner** conforme aula anterior.

# **Modelando os dados da 2a parte do software**



**Vamos criar nosso  
próximo CRUD**

# Modelando os dados da 2a parte do software

| Moedas   |       |                                                                         |
|----------|-------|-------------------------------------------------------------------------|
| Nome     | Sigla | Imagem                                                                  |
| Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   |
| Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> |

# Modelando os dados da 2a parte do software

| Moedas   |       |                                                                         |                   |
|----------|-------|-------------------------------------------------------------------------|-------------------|
| Nome     | Sigla | Imagem                                                                  | Tipo de Mineração |
| Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   | PoW               |
| Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> | PoS               |

# Modelando os dados da 2a parte do software

- **PoW** - Proof of Work (Prova de Trabalho)
- **PoS** - Proof of Stake (Prova de Participação)
- **PoC** - Proof of Capacity (Prova de Capacidade)



# Modelando os dados da 2a parte do software

| Moedas   |       |                                                                         |                   |
|----------|-------|-------------------------------------------------------------------------|-------------------|
| Nome     | Sigla | Imagem                                                                  | Tipo de Mineração |
| Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   | PoW               |
| Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> | Po <b>Z</b>       |



# Modelando os dados da 2a parte do software

- Todo model identifica unicamente seus registros através de um ID adicionado de forma automática e incremental.

| ID | Nome     | Sigla | Imagem                                                                  |
|----|----------|-------|-------------------------------------------------------------------------|
| 1  | Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   |
| 2  | Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> |

# Modelando os dados da 2a parte do software

## Moedas

| ID | Nome     | Sigla | Imagem                       | Tipo de Mineração |
|----|----------|-------|------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | http://site.com/bitcoin.jpg  | 1                 |
| 2  | Ethereum | ETH   | http://site.com/ethereum.jpg | 2                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |



# **Criando o 2o CRUD**

# Criando o 2o CRUD

## Moedas

| ID | Nome     | Sigla | Imagem                       | Tipo de Mineração |
|----|----------|-------|------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | http://site.com/bitcoin.jpg  | 1                 |
| 2  | Ethereum | ETH   | http://site.com/ethereum.jpg | 2                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

# Criando o 2o CRUD

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

- Mining Type
  - name: string
  - acronym: string

```
rails g scaffold MiningType name:string acronym:string
```

## Criando o 2o CRUD

- Após gerar o CRUD, faça a **migração** e ajuste o **link** na página principal da aplicação.
- Aproveite e ajuste também o **db:seed** para cadastrar os tipos de mineração.

# Corrigindo o CRUD



## Corrigindo o CRUD

- No nosso último CRUD geramos o tipo de mineração usando o comando abaixo...

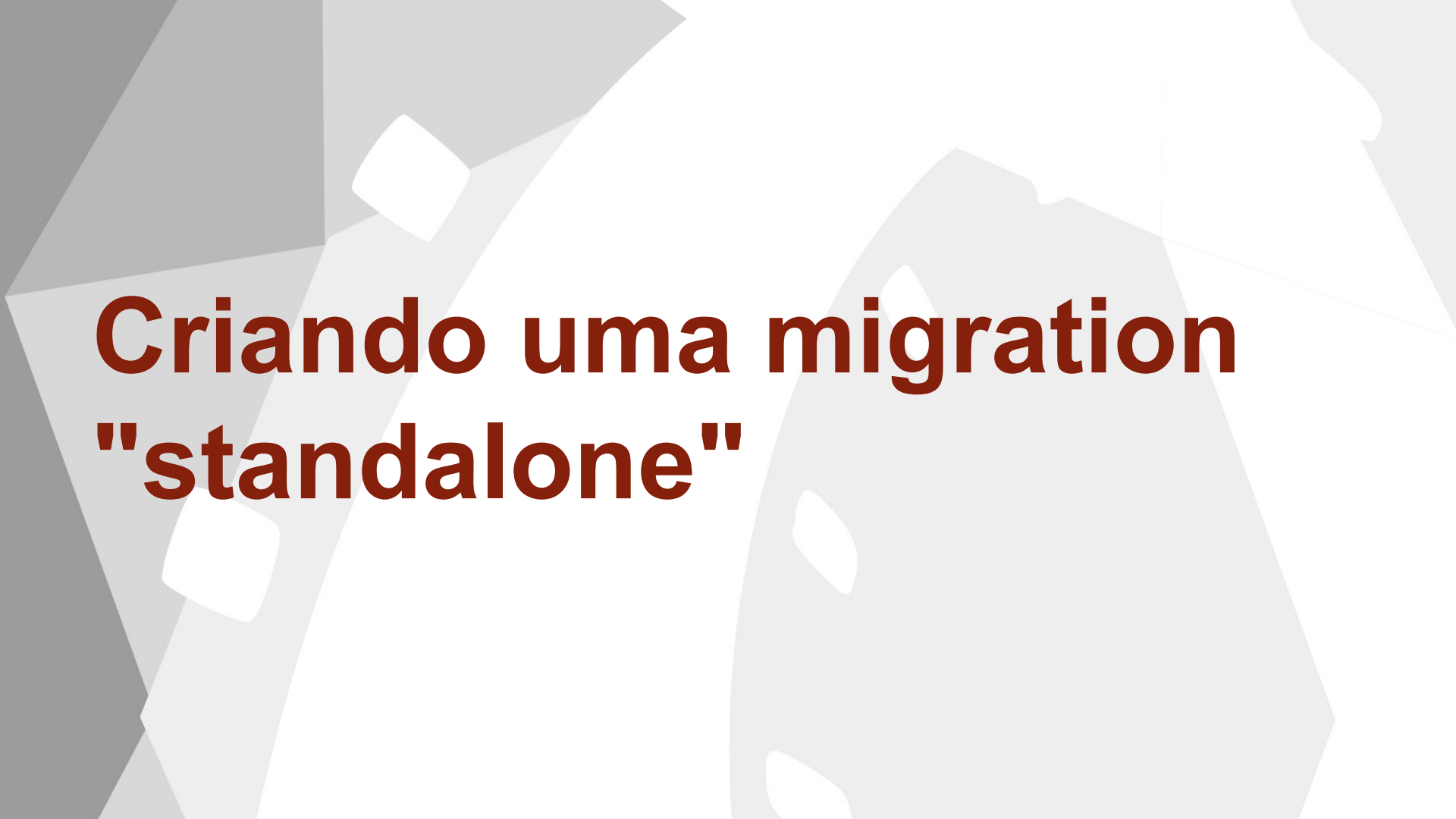
```
rails g scaffold MiningType name:string acronym:string
```

- Só que uma boa prática que acabei me passando foi usar o nome do campo “**name**”. Nesse caso vamos usar “**description**”.

# Corrigindo o CRUD

- Sendo assim temos duas opções, ou alterar em todos os lugares onde o nome do campo “name” aparece, ou, como ainda não fizemos nada além do CRUD, apagar e recriá-lo da forma certa.
- Sendo assim, vamos preferir apagar e recriar por ser mais simples. Veja...

```
rails db:rollback  
rails d scaffold MiningType  
rails g scaffold MiningType description:string acronym:string  
rails db:migrate
```

The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

# **Criando uma migration "standalone"**

# Criando uma migration "standalone"

## Moedas

| ID | Nome     | Sigla | Imagem                                                                  | Tipo de Mineração |
|----|----------|-------|-------------------------------------------------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   | 1                 |
| 2  | Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> | 2                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

# Criando uma migration "standalone"

- Vamos começar criando uma migração para o novo campo que será adicionado na tabela de moedas (coins).
  - [https://guides.rubyonrails.org/active\\_record\\_migrations.html#creating-a-migration](https://guides.rubyonrails.org/active_record_migrations.html#creating-a-migration)

# Criando uma migration "standalone"

- `rails g migration AddMiningTypeToCoins mining_type:references`
- Essa migração vai adicionar um campo na tabela coins que fará um relacionamento/associação com a tabela de tipos de mineração.
- Perceba também que isso não cria nem altera nenhuma view

# Criando uma migration "standalone"

- Após criar a migração você deve executar o `rails db:migrate` para que as alterações sejam aplicadas ao banco de dados.



**Associação  
"belongs\_to"**



# Associação "belongs\_to"

## Moedas

| ID | Nome     | Sigla | Imagem                                                                  | Tipo de Mineração |
|----|----------|-------|-------------------------------------------------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   | 1                 |
| 2  | Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> | 2                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

# Associação "belongs\_to"

## SQLite (Tabelas)

### coins

- id
- description
- acronym
- url\_image
- mining\_type

### mining\_types

- id
- description
- acronym

Por convenção o Rails "exige" que o campo que vai se relacionar com a outra tabela tenha o mesmo nome no singular...

Use o **rails dbconsole** e o **rails console** para inspecionar essa informação...

# Associação "belongs\_to"

## SQLite (Tabelas)

### coins

- id (PK)
- description
- acronym
- url\_image
- mining\_type\_id (FK)

### mining\_types

- id (PK)
- description
- acronym

Percebe-se que existe um **\_id** no campo de referência... ficando **mining\_type\_id** para facilitar ainda mais o entendimento quando estamos programando.

Isso também é uma convenção!

## Associação "belongs\_to"

- Agora que já resolvemos o "lado das tabelas" precisamos também deixar o model funcionando nesse formato.
- Para isso, adicione a associação ao model Coin **belongs\_to :mining\_type**.

```
class Coin < ApplicationRecord
  belongs_to :mining_type
end
```

# Associação "belongs\_to"

- Isso quer dizer que os registros desse model Coin estarão associados (pertencendo a) a um registro do model Mining Type
  - [https://guides.rubyonrails.org/association\\_basics.html#the-belongs-to-association](https://guides.rubyonrails.org/association_basics.html#the-belongs-to-association)

# Associação "belongs\_to"

## SQLite (Tabelas)

### coins

- id
- description
- acronym
- url\_image
- mining\_type\_id (FK)

### mining\_types

- id
- description
- acronym

## Models

```
class Coin < ApplicationRecord
  belongs_to :mining_type
end
```

```
class MiningType < ApplicationRecord
end
```

# Associação "belongs\_to"

- Faça testes no **rails console**
  - `c = Coin.first`
  - `m = MiningType.first`
  - `c.mining_type = m`
  - `c.save!`



**Associação  
"has\_many"**



# Associação "belongs\_to"

## Moedas

| ID | Nome     | Sigla | Imagem                                                                  | Tipo de Mineração |
|----|----------|-------|-------------------------------------------------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | <a href="http://site.com/bitcoin.jpg">http://site.com/bitcoin.jpg</a>   | 1                 |
| 2  | Ethereum | ETH   | <a href="http://site.com/ethereum.jpg">http://site.com/ethereum.jpg</a> | 1                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

# Associação "has\_many"

## Moedas

| ID | Nome     | Sigla | Imagem                       | Tipo de Mineração |
|----|----------|-------|------------------------------|-------------------|
| 1  | Bitcoin  | BTC   | http://site.com/bitcoin.jpg  | 1                 |
| 2  | Ethereum | ETH   | http://site.com/ethereum.jpg | 1                 |

## Tipo de Mineração

| ID | Nome           | Sigla |
|----|----------------|-------|
| 1  | Proof of Work  | PoW   |
| 2  | Proof of Stake | PoS   |

## Associação "has\_many"

- Percebemos que tudo já está pronto em nossa tabela, faltando apenas indicar ao model `MiningType` que queremos efetuar essa associação.
- Sendo assim, basta adicionar **`has_many :coins`** no model `MiningType`.
  - `https://guides.rubyonrails.org/association\_basics.html#the-has-many-association`

# Associação "has\_many"

- Faça testes no **rails console**
  - `c = Coin.last`
  - `m = MiningType.last`
  - `m.coins << c`
  - `m.save!`

# Ajustando a Task

## “dev:add\_coins”

## Ajustando a Task dev:add\_coins

- Agora que temos as associações mapeadas e funcionando em nossa aplicação, devemos ajustar a Task **dev:add\_coins**, para isso complemente as moedas como o modelo abaixo:

- {
- description: "Bitcoin",
- acronym: "BTC",
- url\_image: "https://assets..."
- **mining\_type: MiningType.all.sample**
- }

# Conhecendo os métodos `.map` e o `.pluck`

# O método .map

- <https://ruby-doc.org/core-2.2.0/Array.html#method-i-map>
- Imagine que você tem um array e quer transformar ele em outro... ex:
  - `[1,2,3,4,5] => [2,4,6,8,10]`
- Perceba que o segundo é baseado no primeiro.
- Poderíamos fazer uma iteração no primeiro para obter o segundo facilmente, mas, o ruby possui um método que permite a iteração de uma coleção retornando um novo array no final.



# O método .map

- O método map funciona assim...
  - `[1,2,3,4,5].map do |i|`
    - `i * 2`
  - `end`
- ou se preferir...
  - `[1,2,3,4,5].map { |i| i * 2 }`
- A saída será um novo array baseado no primeiro.
  - `[2,4,6,8,10]`

## O método .map

- Ainda sobre o método .map, podemos usá-lo em conjunto com hashes ou resultados no ActiveRecord, veja:
- `Coin.all.map { |coin| coin.description }`
- Uma forma de melhorar isso é usando o "ampersand" **&**.
- `Coin.all.map(&:description)`

## O método .pluck

- [https://guides.rubyonrails.org/active\\_record\\_querying.html#pluck](https://guides.rubyonrails.org/active_record_querying.html#pluck)
- Por fim, temos o método pluck, que funciona de forma parecida com o map, mas que reduz ainda mais o esforço na hora de obter elementos determinados resultados em formato de Array, a partir de resultados do ActiveRecord. Veja
- `Coin.all.pluck(:description)`

# Conhecendo o helper "select"

# Conhecendo o helper "select"

- Chegou a hora de ajustarmos a view para que seja possível selecionar o tipo de mineração, e para isso usaremos o helper **select**.
- <https://api.rubyonrails.org/v5.2.0/classes/ActionView/Helpers/FormOptionsHelper.html#method-i-select>

# Conhecendo o helper "select"

- Deixe o código da **views/coins/\_form.html.erb** assim:

```
<div class="field">
  <%= form.label :mining_type_id %>
  <%= form.select("mining_type_id", MiningType.all.collect
{|m| [ m.description, m.id ] }, {include_blank:
'Selecione...'})
%>
</div>
```

# Conhecendo o helper "select"

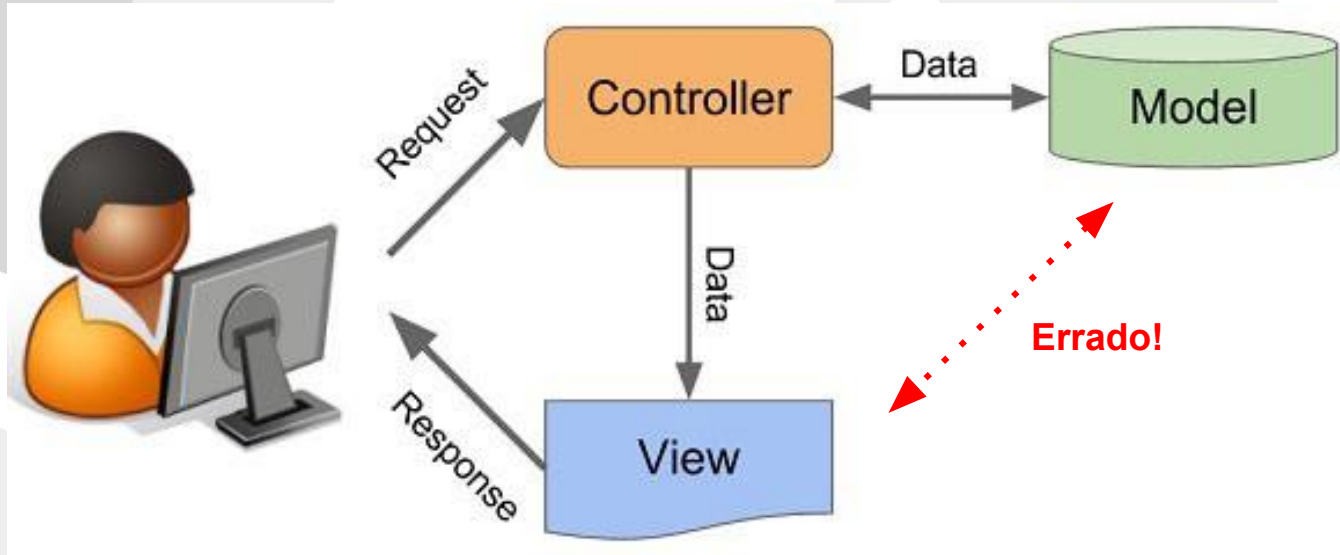
- Por fim, não esqueça de "liberar" o parâmetro no **coins\_controller.br**

```
def coin_params
  params.require(:coin).permit(:description, :acronym,
:url_image, :mining_type_id)
end
```

# **Padronizando o "select" ao MVC**



# Padronizando o "select" ao MVC

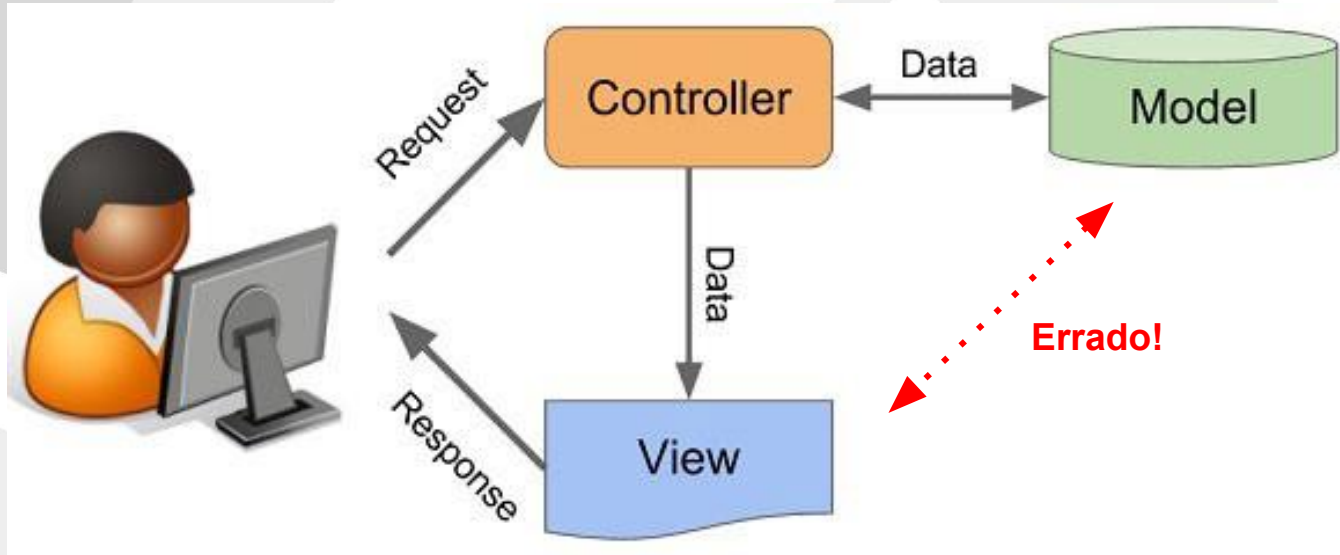


# Padronizando o "select" ao MVC

- A primeira refatoração que podemos fazer na view é usar o pluck

```
<%= form.select("mining_type_id",  
MiningType.all.pluck(:description, :id), {include_blank:  
'Selecione...'})  
%>
```

# Padronizando o "select" ao MVC



# Padronizando o "select" ao MVC

- Mas só usando o pluck ainda estamos "ferindo o MVC", então vamos "migrar" esse código para o controller, mais especificamente criando um método privado chamado **set\_mining\_type\_options**

```
def set_mining_type_options
  @mining_type_options = MiningType.all.pluck(:description,
:id)
end
```

# Padronizando o "select" ao MVC

- Em seguida, crie um filtro para executar o método no **new**, e no **edit**
- `before_action :set_mining_type_options, only: [:new, :edit]`

# Conhecendo o helper "select"

- Agora basta usar o @mining\_type\_options como opção do select...
- ```
<%= form.select("mining_type_id", @mining_type_options, {include_blank: 'Selecione...'}) %>
```
- Pronto! Agora estamos seguindo a arquitetura MVC!

# **Um pouco sobre arquivos YAML**

# Um pouco sobre arquivos YAML

- "YAML é um formato de serialização (codificação de dados) de dados legíveis por humanos"
- <https://pt.wikipedia.org/wiki/YAML>



# Um pouco sobre arquivos YAML

- Veja esse exemplo:

**Endereço:**

**Rua Abelardo, número 09**

**CEP: 00.000-000**

# Um pouco sobre arquivos YAML

- Veja esse exemplo:

Endereço:  
Rua Abelardo, número 09  
CEP: 00.000-000

```
<endereco>  
  <rua>  
    Abelardo  
  </rua>  
  <numero>  
    09  
  </numero>  
  <cep>  
    00.000-000  
  </cep>  
</endereco>
```

**XML**

# Um pouco sobre arquivos YAML

- Veja esse exemplo:

Endereço:  
Rua Abelardo, número 09  
CEP: 00.000-000

```
<endereco>  
  <rua>  
    Abelardo  
  </rua>  
  <numero>  
    09  
  </numero>  
  <cep>  
    00.000-000  
  </cep>  
</endereco>
```

```
endereco:  
  rua: Abelardo  
  numero: 09  
  cep: 00.000-00
```

YAML

# Um pouco sobre arquivos YAML

- Ao usar YAML precisamos ter atenção na indentação do que é escrito, ou seja, o YAML se baseia no espaçamento e quebra de linhas.

```
endereco:  
  rua: Abelardo  
  numero: 09  
  cep: 00.000-00
```

YAML

The background features a light gray abstract design with various geometric shapes, including triangles and polygons. A faint, stylized outline of a globe is visible in the upper right corner. The text is centered in a bold, dark red font.

**Ativando o i18n**

# Ativando o i18n

- **I18n** é a "sigla" para **Internationalization**
- `https://guides.rubyonrails.org/i18n.html`

## Ativando o i18n

- A primeira coisa que vamos fazer é, adicionar a gem **'rails-i18n'** no **Gemfile** e na sequência informar ao Rails quais localidades devem estar disponíveis na aplicação.
- Para isso, vamos na pasta **config/initializers**, onde devemos criar um arquivo **locale.rb** e adicionar o seguinte conteúdo:

```
I18n.available_locales = [:en, 'pt-BR']  
I18n.default_locale = :en
```

## Ativando o i18n

- Após a configuração, você pode usar pelo menos 3 métodos para identificar qual a localidade que a aplicação está atualmente e quais as localidades disponíveis.
- `I18n.available_locales`
- `I18n.locale`
- `I18n.default_locale`



## Ativando o i18n

- Aproveite e crie um helper para identificar qual idioma está sendo usado na aplicação.
- ```
def locale
```

  - ```
I18n.locale == :en ? "Estados Unidos" :
```

```
"Português do Brasil"
```
- ```
end
```

The background features a light gray globe with white latitude and longitude lines. Overlaid on the globe are several large, semi-transparent geometric shapes in shades of gray, including triangles and polygons. There are also a few small, white, rounded rectangular shapes scattered across the composition.

**Usando o i18n**

## Usando o `i18n`

- Em regras gerais, temos 2 métodos que podem fazer o uso do `i18n`.
- O método `I18n.t()` e o método `I18n.l()`

O primeiro vem de **translate**, onde informamos uma chave e recebemos uma tradução, baseada em arquivos YAML que ficam na pasta **config/locales**. Vamos começar usando ele.

# Usando o i18n

- Para um teste, coloque em sua view `I18n.t('hello')` isso mostrará como saída na view **"Hello World"**.
- Agora pare a aplicação, vá em **`config/initializers/locale.rb`** e deixe como default o `pt-BR`.

```
I18n.default_locale = 'pt-BR'
```

# Usando o i18n

- Na sequência crie um arquivo pt-br.yml e adicione o conteúdo:
- "pt-BR":
  - hello: "Olá Mundo!"
- Inicie novamente a aplicação e veja agora que o texto mostrado é **"Olá Mundo!"**

## Usando o `i18n`

- Por fim, podemos usar o método `I18n.l()`, que vem de "localize" e tem a função de deixar datas e horas no formato da localidade atual.
- Faça um teste na sua view `I18n.l(Date.today)` e veja que o resultado é a data já no formato brasileiro.

The background features a light gray abstract design with overlapping geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the upper right corner. The text is centered in a bold, dark red font.

**i18n para Models**

# i18n para Models

- i18n para models
- `https://guides.rubyonrails.org/i18n.html#translations-for-active-record-models`

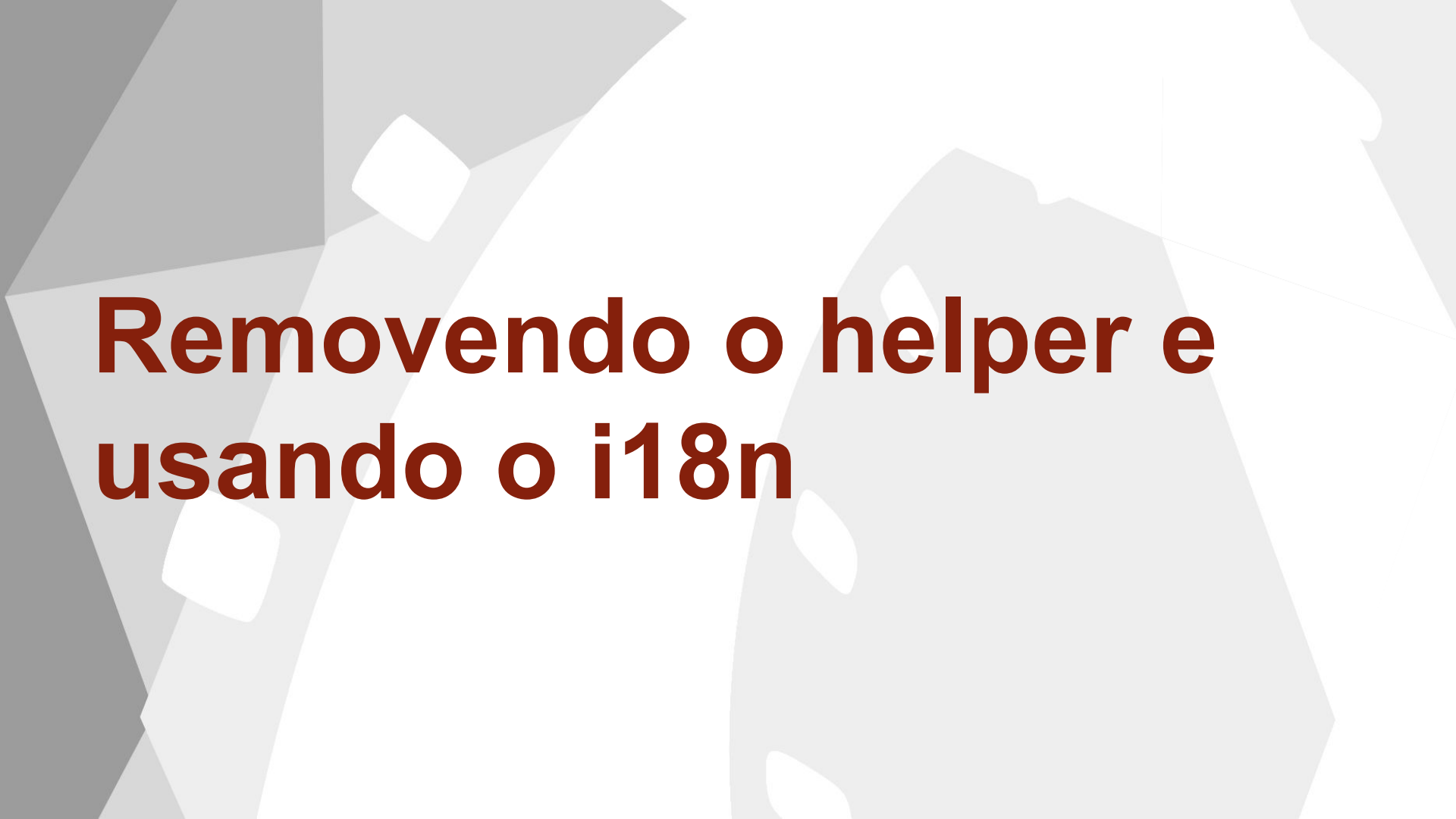


# i18n para Models

- Para usar i18n nos models basta criar um arquivo **.yml** em **config/locales** com a seguinte estrutura:
  - 'pt-BR':
    - activerecord:
      - models:
        - coin:
          - one: Moeda
          - other: Moedas
        - attributes:
          - coin:
            - description: "Descrição"

# i18n para Models

- Após isso todos os **labels** dos formulários serão traduzidos de forma automática.
- Se quiser forçar uma tradução use...
- `Model.human_attribute_name(attribute)`



**Removendo o helper e  
usando o i18n**

# Removendo o helper e usando o i18n

- Percebi que podemos usar o próprio sistema de i18n ao invés do helper que criamos para mostrar o idioma atual. Para isso, crie a chave no arquivo **config/locales/pt-BR.yml**
- `actual_locale: Português do Brasil`
- Em seguida faça o mesmo para o idioma em inglês no arquivo **config/locales/en.yml**.
- `actual_locale: Inglês`

# Removendo o helper e usando o i18n

- Por fim, substitua a chamada do helper para:
- `<%= t('actual_locale') %>`

# Cookies e Sessions

The background features a series of overlapping, semi-transparent geometric shapes in shades of gray and white. Scattered across these shapes are several stylized, light gray illustrations of cookies, some with white icing decorations.

# Stateless vs Stateful

Conexões HTTP são **stateless**, ou seja, **cada requisição é única** e o servidor nunca vai saber quem fez a requisição e o que ocorreu depois dela.

Caso o HTTP fosse **stateful**, ele **manteria o estado** entre as requisições, ou seja, o servidor saberia o “histórico” da requisição.

Para resolver o problema de ser stateless, podemos usar algumas soluções.

# Cookies

- Podemos usar cookies para armazenar dados no navegador, que podem ser persistidos entre requisições, para, por exemplo, um próximo retorno do usuário ao site.
- <https://api.rubyonrails.org/v5.2.1/classes/ActionDispatch/Cookies.html>  

```
cookies[:user_name] = "Jackson Pires"
```



# Sessions

- Sessions são muito parecidos com cookies, no entanto os dados são armazenados no servidor.
- <https://guides.rubyonrails.org/security.html>

```
session[:user_name] = "Jackson Pires"
```

The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

**Usando a aplicação  
com vários idiomas em  
tempo de execução**

# Vários idiomas

- Uma das formas de se usar vários idiomas é através de parâmetros de URL
- `https://guides.rubyonrails.org/i18n.html`

# Vários idiomas

- No **ApplicationController** faça:

```
before_action :set_locale
def set_locale
  if params[:locale]
    cookies[:locale] = params[:locale]
  end
  if cookies[:locale]
    if I18n.locale != cookies[:locale]
      I18n.locale = cookies[:locale]
    end
  end
end
```

# Vários idiomas

- Na view, crie dois links para os idiomas:

```
<%= link_to 'Português', '/?locale=pt-BR' %>  
<%= link_to 'Inglês', '/?locale=en' %>
```

ou

```
<%= link_to 'Português', root_path(locale: 'pt-BR') %>  
<%= link_to 'Inglês', root_path(locale: 'en') %>
```



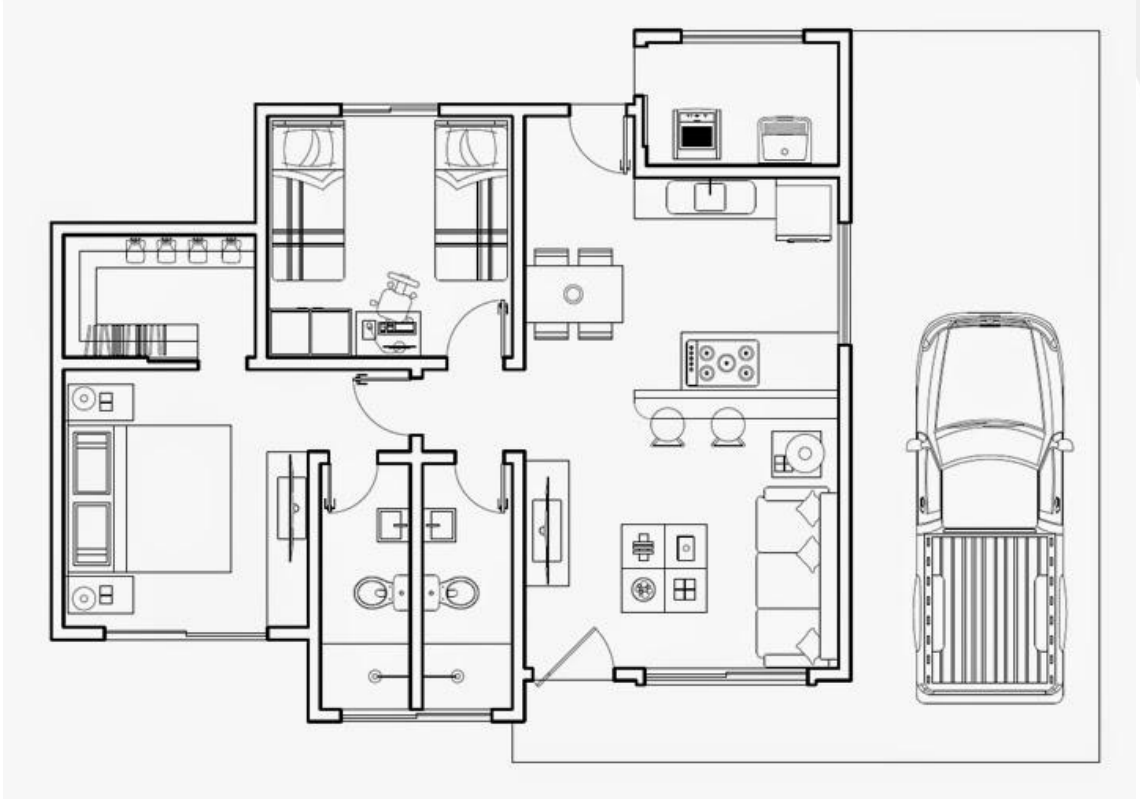
# **Entendendo Javascript em 1 aula!**

# JS em 1 aula

- Em desenvolvimento Web é praticamente impossível fazer algo que não envolva HTML, CSS e Javascript.
- Indo por partes, **HTML** (Hyper Text Markup Language), é uma linguagem de marcação que permite estruturar uma página Web, indicando de forma semântica o que é o cabeçalho, rodapé, corpo e outras partes de uma página web.

# JS em 1 aula

- A grosso modo podemos dizer que o HTML é o projeto arquitetônico





# JS em 1 aula

- Exemplo:

```
<html>
  <head></head>
  <body>
    <p> Olá! </p>
  </body>
</html>
```

# JS em 1 aula

- Quando falamos em CSS estamos falando, a grosso modo, em dar um visual ao nosso HTML.



# JS em 1 aula

- Exemplo:

```
<html>
  <head></head>
  <body>
    <p style="color: red"> Olá! </p>
  </body>
</html>
```

# JS em 1 aula

- Quando o HTML + CSS é renderizado pelo navegador, temos o resultado da página web.



# JS em 1 aula

- E onde entra o Javascript?
- Bem, o JS fará o funcionamento das coisas **dinâmicas** do site.
- O JS, a princípio, é uma linguagem que só funciona no navegador.
- Uma das coisas mais importantes que o JS faz é permitir adicionar, remover e alterar o HTML enquanto é exibido pelo navegador.

# JS em 1 aula

- Ou seja, só vamos conseguir páginas dinâmicas usando Javascript em algum momento.



# JS em 1 aula

- Exemplo:

```
<html>
  <head></head>
  <body>
    <p id="p1" style="color: red"> Olá! </p>

    <script>
      document.getElementById("p1").innerHTML = "Hello!";
    </script>
  </body>
</html>
```

The background features a light gray abstract design with various geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the upper right corner. The text is centered and rendered in a bold, dark red font.

# **Conhecendo o Asset Pipeline**



# Conhecendo o Asset Pipeline

- Asset Pipeline

`https://guides.rubyonrails.org/asset\_pipeline.html`

The background features a complex geometric design. On the left, there are several overlapping triangles in shades of gray. A large, light gray shape resembling a gear or a stylized letter 'A' dominates the center and right. This shape has several white, pill-shaped cutouts. The overall color palette is monochromatic, consisting of various tones of gray and white, with the text providing a contrasting dark red color.

# **Asset Pipeline**

# Conhecendo o Asset Pipeline

- Quando falamos de “assets” em desenvolvimento web estamos na maioria das vezes falando sobre **imagens**, **CSS** e **Javascript**.
- Veja a pasta **app/assets** da sua aplicação. Não por acaso ela possui as pastas **images**, **stylesheets** e **javascripts**.

# Conhecendo o Asset Pipeline

- O asset pipeline do Rails permite concatenar, minificar ou comprimir assets CSS e Javascript gerando ao final apenas um arquivo para **diminuir a quantidade de requisições que o navegador faz ao servidor**.
- A ideia de “asset pipeline” é fazer os assets passarem por “vários estágios/etapas” (minificar, concatenar, etc) até atingir um único asset final.

# Conhecendo o Asset Pipeline

- O asset pipeline na verdade é uma gem chamada **sprockets-rails** que foi unificado ao Rails, ou seja, era um projeto externo.



**Fingerprint**

# Fingerprint

- Todo arquivo CSS / JS carregado no navegador está propício a cache, visto que quanto menos requisições forem feitas ao servidor, melhor.
- No entanto, quando estamos desenvolvendo a aplicação, é normal que alteremos os assets com certa frequência e isso pode acabar “confundindo” o navegador em relação a fazer o cache.

# Fingerprint

- Sendo assim, o Rails utiliza uma técnica de “fingerprint” para contornar esse problema.
- Essa técnica consiste basicamente em fazer com que o nome do arquivo seja alterado a cada alteração no mesmo e isso por si só já evita o cache do navegador.





**ExecJS**

# ExecJS

- O ExecJS basicamente é um “runtime” do javascript que é exigido pelo Rails para que o Asset Pipeline funcione, visto que as ferramentas que concatenam e minificam os assets em sua grande maioria são desenvolvidas em Javascript.
  - **therubyracer**
  - **nodejs**

The background features a light gray world map centered on the Atlantic Ocean. Overlaid on the map are several large, semi-transparent geometric shapes in shades of gray and white, including triangles and polygons, creating a modern, abstract design.

# Organização

# Organização

- Os assets devem ficar em pastas específicas.
  - **app/assets:** Para assets criados pelo próprio Rails
  - **lib/assets:** Para assets que você mesmo criou
  - **vendor/assets:** Para assets que você “pegou” de terceiros

# **Pré-compilando e isolando assets por controller**

# Isolando assets por controller

- Usando o Asset Pipeline

`https://guides.rubyonrails.org/asset\_pipeline.html#controller-specific-assets`

# Isolando assets por controller

- Podemos usar o parâmetro **params[:controller]** para identificar qual controller está sendo invocado na request.
- Dessa forma podemos aproveitar e isolar CSS e JS para serem carregados de acordo com o controller. Veja o exemplo:

```
<% stylesheet_link_tag params[:controller] %>
```

# **Pré-compilando Assets**



# Pré-Compilando Assets

- Pré-Compilando Assets

`https://guides.rubyonrails.org/asset\_pipeline.html#precompiling-assets`

# Pré-Compilando Assets

- A pré-compilação é exigida para arquivos que sejam usados explicitamente além o **application.js** e **application.css**.
- Nesse caso, precisamos ir em **config/initializers/assets.rb** e adicionar manualmente os assets extras que vamos usar.
- No caso dos controllers, precisaremos fazer essa alteração para que eles sejam conhecidos.



# Usando o Asset Pipeline

# Usando o Asset Pipeline

- Usando o Asset Pipeline

`https://guides.rubyonrails.org/asset\_pipeline.html#coding-links-to-assets`



**ERB + CSS**

# ERB + CSS

- Para usar código ruby misturado ao seu CSS, basta que você altere a extensão do arquivo css/scss para **css.erb** ou **scss.erb**.
- Para nosso exemplo, altere o arquivo **app/stylesheets/scaffolds.css.erb**

```
body {  
  background-color: <%= Rails.env.development? ? 'white' : 'yellow' %>;  
}
```



**SASS**

# SASS

- Como falado anteriormente, o SASS adiciona superpoderes ao seu CSS, mas, uma das coisas que mesmo vc não sendo do front-end deve aprender a usar é a tag para referir-se aos assets. Nesse caso, uma imagem, por exemplo.
- Comece alterando o arquivo para **scaffolds.scss.erb** e baixando uma imagem e colocando-a em **lib/images**.



# SASS

- Na sequência, altere o arquivo **scaffolds.css** para...

```
body {  
  background-image: asset-url("imagem.png");  
}
```



**ERB + JS**

# ERB + JS

- Para usar código ruby misturado ao seu Javascript, basta que você altere a extensão do arquivo **js** para **.js.erb**
- Para nosso exemplo, altere o arquivo **app/javascripts/application.js.erb**

```
<% msg = Time.now.hour < 12 ? "Bom dia!" : "Olá!!!" %>  
alert("<%= msg %>");
```

# Usando tasks para pré-compilar assets

# Usando tasks para pré-compilar assets

- `https://guides.rubyonrails.org/asset\_pipeline.html#precompiling-assets`
- Comece digitando **rails -T assets** para verificar as possíveis tasks relacionadas aos assets.

# Usando tasks para pré-compilar assets

- Podemos observar que entre elas temos a **assets:precompile** que é uma task que vai pré-compilar os assets e disponibilizá-los na pasta **public/assets**
- Essa pré-compilação só é necessário em produção, visto que em desenvolvimento a compilação é transparente para o desenvolvedor

# Usando tasks para pré-compilar assets

- Sendo assim, para fazer um teste rode.
  - **rails assets:precompile**
- Após o comando finalizar, observe a pasta **public/assets**, bem como o arquivo **.sprockets-manifest\*** que contém a referência para todos os arquivos.

# Usando tasks para pré-compilar assets

- Como a pré-compilação só é necessária em produção, vamos usar uma task para remover os assets pré-compilados. Rode **rails assets:clobber**



# Usando uma biblioteca JS

# Usando uma biblioteca JS

- No Rails, podemos usar uma biblioteca JS pelo menos de 3 formas diferentes.
- A primeira é baixando a biblioteca colocando na pastas **vendor/assets**
- Vamos fazer um teste com a biblioteca Notify JS
  - `https://notifyjs.jpillora.com/`

# Usando uma biblioteca JS

- Baixe os arquivos (notify.js e jquery.js - cdn) e disponibilize em vendor/assets.
- Adicione os dois assets no **config/initializers/assets.rb**
- Adicione as chamadas do javascript no layout **application.html.erb**
- Adicione também no layout application o script...

```
<script> $.notify("Funcionou!!!", "success"); </script>
```



**Usando uma biblioteca  
JS ([rails-assets.org](https://rails-assets.org))**

## Usando uma biblioteca JS ([rails-assets.org](https://rails-assets.org))

- Uma outra forma de usar bibliotecas JS é usar o projeto rails-assets.org
- Esse site se propõe a "transformar" bibliotecas JS em gems, facilitando ainda mais o uso.
- Vamos usar esse projeto para a biblioteca Notify JS

# Usando uma biblioteca JS ([rails-assets.org](https://rails-assets.org))

- Comece removendo a biblioteca Notify JS de **vendor/assets**
- Em seguida, remova a pré-compilação do **config/initializers/assets.rb**
- Por fim, remova a entrada do jquery e do notify do **app/views/layouts/application.html**

## Usando uma biblioteca JS ([rails-assets.org](https://rails-assets.org))

- Agora, acesse o site [rails-assets.org](https://rails-assets.org)
- Procure pela biblioteca Notify JS e JQuery
- Verifique que para instalar basta adicionar a entrada no Gemfile, rodar o bundler e adicionar a entrada da biblioteca em application JS, conforme o site indica.

# Usando uma biblioteca JS (rails-assets.org)

- `#Gemfile`
  - `gem 'rails-assets-jquery', source: 'https://rails-assets.org'`
  - `gem 'rails-assets-notifyjs', source: 'https://rails-assets.org'`
- 
- `#application.js`
  - `//= require jquery`
  - `//= require notifyjs`



# Usando uma biblioteca JS ([rails-assets.org](https://rails-assets.org))

- Inicie a aplicação e verifique que tudo continua a funcionar.

# Conhecendo o Yarn

# Conhecendo o Yarn

- Yarn

`https://yarnpkg.com/pt-BR/`

O Yarn é um gerenciador de pacotes para bibliotecas Javascript adotado pela comunidade Rails na versão 5.1. Ele facilita bastante quando queremos usar uma biblioteca JS, evitando que precisemos informar ao asset pipeline os arquivos que devem ser pré-compilados, bastando apenas carregar e usar a biblioteca

# Conhecendo o Yarn

- Vamos começar Instalando Yarn

```
npm i --global yarn
```

# Usando uma biblioteca JS (Yarn)

# Usando uma biblioteca JS (Yarn)

*Atenção: o Yarn NÃO é um projeto que o time de desenvolvimento Ruby on Rails mantém, eles apenas recomendam o uso e dão a possibilidade de se usar o yarn em projetos Rails.*

- Para utilizá-lo, a primeira coisa que vamos fazer é remover as gems do rails-assets do Gemfile e remover a entrada do application.js. Rode o bundle após remover.

# Usando uma biblioteca JS (Yarn)

- Agora vamos iniciar o Yarn para a aplicação Rails. Para isso rode **yarn init**
- Perceba que foi o arquivo **package.json**
- Agora basta procurar o nome da biblioteca no site **yarnpkg.com** , adicioná-lo via yarn e depois carregar a biblioteca no **application.js**

# Usando uma biblioteca JS (Yarn)

```
yarn add jquery
```

```
yarn add notify-js-legacy
```

- Verifique o conteúdo da pasta **node\_modules**
- Verifique também o conteúdo do arquivo **package.json**

```
#application.js
```

```
//=require jquery
```

```
//=require notify-js-legacy/notify
```



# Usando uma biblioteca JS (Yarn)

- Lembre-se:

Gemfile → Gemfile.lock

package.json → yarn.lock

# Usando uma biblioteca JS (Yarn)

- É comum que a pasta **node\_modules** não fique armazenada no projeto do github pois a partir do package.json podemos instalar todas as bibliotecas novamente. Para isso rode na raiz do projeto:
- `yarn install`

# **Primeiros passos com o Bootstrap**

# Primeiros passos com o Bootstrap

- Bootstrap

`http://getbootstrap.com/`

Bootstrap é um kit de ferramentas para desenvolver com HTML, CSS e JS.

# Primeiros passos com o Bootstrap

- Instalando

```
yarn add bootstrap  
yarn add popper.js
```

```
#application.js  
//= require bootstrap/dist/js/bootstrap  
//= require popper.js/dist/popper
```

```
#application.css  
*= require bootstrap/dist/css/bootstrap
```

# Primeiros passos com o Bootstrap

- Testando...
  - <http://getbootstrap.com/docs/4.1/content/tables/>
  - Adicione a classe **"table"** na tabela das moedas e veja o que ocorre.
    - `<table class="table table-hover"> ... </table>`

# Melhorando a Welcome Page

# Melhorando a Welcome Page

Antes de mais nada, remova o **scaffolds.css** do **layouts/application.html.erb**



# Melhorando a Welcome Page

- Em seguida vamos fazer adicionar uma **div** com uma classe **container-fluid**, no layouts/application.html.erb para que tudo fique "ajustável".
- <http://getbootstrap.com/docs/4.1/layout/overview/>

```
<body>
  <div class="container-fluid">
    <%= yield %>
  </div>
</body>
```

# Melhorando a Welcome Page

- O próximo passo é colocar um **jumbotron** na index.
- <http://getbootstrap.com/docs/4.1/components/jumbotron/>

```
<div class="jumbotron">
  <h1 class="display-4">Seja Bem-Vindo!</h1>
  <p class="lead">Esse é o Crypto Wallet, um app para você cadastrar e
acompanhar tudo sobre as suas criptomoedas.</p>
  <hr class="my-4">
  <p>Para começar, selecione uma das opções abaixo:</p>
  <%= render "menu" %>
</div>
```

# Melhorando a Welcome Page

- Agora ajuste a **partial** do menu.

```
<%= link_to "Moedas", coins_path, class:"btn btn-primary btn-lg",  
role:"button" %>  
<%= link_to "Tipos de Mineração", mining_types_path, class:"btn btn-primary  
btn-lg", role:"button" %>
```

# Melhorando a Welcome Page

- Adicione uma navegação para a tradução do site...

```
<div class="container-fluid">
  <ul class="nav justify-content-end">
    <li class="nav-item">
      <%= link_to 'Português', root_path(locale: 'pt-BR'),
class:"nav-link" %>
    </li>
    <li class="nav-item">
      <%= link_to 'English', root_path(locale: 'en'), class:"nav-link"
%>
    </li>
  </ul>
  <%= yield %>
</div>
```

# **Arrumando o i18n da Welcome Page**

# Arrumando o i18n da Welcome Page

Vamos começar alterando o **views/welcome/index.html.erb** para:

```
<div class="jumbotron">
  <h1 class="display-4"><%= t('welcome') %></h1>
  <p class="lead"><%= t('message.welcome') %></p>
  <hr class="my-4">
  <p><%= t('message.menu') %></p>
  <%= render "menu" %>
</div>
```

# Arrumando o i18n da Welcome Page

O segundo passo é alterar o **views/welcome/\_menu.html.erb** para:

```
<%= link_to "#{Coin.model_name.human}", coins_path, class:"btn btn-primary btn-lg", role:"button" %>
```

```
<%= link_to "#{MiningType.model_name.human}", mining_types_path, class:"btn btn-primary btn-lg", role:"button" %>
```

# Arrumando o i18n da Welcome Page

Por fim altere os locais **config/locales/pt-BR.yml** para:

```
current_locale: "Português do Brasil"
welcome: "Seja Bem-Vindo!"
message:
  welcome: "Este é o Crypto Wallet, uma app para você cadastrar e
acompanhar as suas criptomoedas."
  menu: "Para começar, escolha uma opção abaixo:"
```



# Arrumando o i18n da Welcome Page

Por fim altere os locales **config/locales/en.yml** para:

```
current_locale: "English (International)"  
welcome: "Welcome!"  
message:  
  welcome: "This is the Crypto Wallet, an app that you can register and  
keep up with your cryptocurrencies."  
  menu: "To start, choose an option below:"
```

# **Melhorando a Index (Coin e MiningType)**

# Melhorando a Index (Coin e MiningType)

Plural para a i18n:

`https://guides.rubyonrails.org/i18n.html#translations-for-active-record-models`

# Melhorando a Index (Coin e MiningType)

Altere a **views/mining\_types/index.html.erb** para:

```
<p id="notice"><%= notice %></p>

<h1><%= Coin.model_name.human(count: @coins.count) %></h1>
<table class="table table-hover">
  ...
  <td><%= link_to t('links.delete'), coin, method: :delete, data: { confirm:
t('messages.confirm') } %></td>
  ...
  <%= link_to t('links.new'), new_coin_path, class: "btn btn-primary" %>
```

# Melhorando a Index (Coin e MiningType)

Altere a **views/mining\_types/index.html.erb** para:

```
<p id="notice"><%= notice %></p>

<h1><%= MiningType.model_name.human(count: @mining_types.count) %></h1>
<table class="table table-hover">
  ...
  <td><%= link_to t('links.delete'), coin, method: :delete, data: { confirm:
t('messages.confirm') } %></td>
  ...
  <%= link_to t('links.new'), new_coin_path, class: "btn btn-primary" %>
```

# Melhorando a Index (Coin e MiningType)

Lembre-se de adicionar a chave **link.new** e **messages.confirm** em **config/locales** para EN e PT-BR

**Adicionando mais i18n  
para Coin e MiningType**

# Adicionando mais i18n para Coin e MiningType

Ajuste o **config/locales/pt-BR.yml** para:

```
links:
  edit: "Editar"
  show: "Mostrar"
  delete: "Apagar"
  new: "Novo"
  back: "Voltar"
new:
  coin: "Nova Moeda"
  mining_type: "Novo Tipo de Mineração"
editing:
  coin: "Editando Moeda"
  mining_type: "Editando Tipo de Mineração"
showing:
  coin: "Mostrando Moeda"
  mining_type: "Mostrando Tipo de Mineração"
```



# Adicionando mais i18n para Coin e MiningType

Ajuste o **config/locales/en.yml** para:

```
links:
  edit: "Edit"
  show: "Show"
  delete: "Delete"
  new: "New"
  back: "Back"
new:
  coin: "New Coin"
  mining_type: "New Mining Type"
editing:
  coin: "Editing Coin"
  mining_type: "Editing Mining Type"
showing:
  coin: "Showing Coin"
  mining_type: "Showing Mining Type"
```

## Adicionando mais i18n para Coin e MiningType

Ajuste o **index** de **coins** e **mining\_types** para:

```
<%= link_to t(new.coin'), new_coin_path, class: "btn btn-primary" %>
```

```
<%= link_to t(new.mining_type), new_coin_path, class: "btn btn-primary" %>
```

# **Melhorando o new e edit (Coin)**

# Melhorando o new e edit (Coin)

Verifique a parte de formulários do bootstrap

>> <http://getbootstrap.com/docs/4.1/components/forms/>

# Melhorando o new e edit (Coin)

Use as classes para construir o formulário

```
class="form-group" e class="form-control"
```

# Melhorando o new e edit (MiningType)

# Melhorando o new e edit (MiningType)

Verifique a parte de formulários do bootstrap

>> <http://getbootstrap.com/docs/4.1/components/forms/>

# Melhorando o new e edit (MiningType)

Use as classes para construir o formulário

```
class="form-group" e class="form-control"
```



# **Melhorando o show (Coin e MiningType)**

# Melhorando o show (Coin e MiningType)

Verifique que é possível fazer formulários desativados

`http://getbootstrap.com/docs/4.1/components/forms/#disabled-forms`

# Melhorando o show (Coin e MiningType)

Altere o formulário usando as novas tags

## **text\_field\_tag**

[https://apidock.com/rails/v4.2.7/ActionView/Helpers/FormTagHelper/text\\_field\\_tag](https://apidock.com/rails/v4.2.7/ActionView/Helpers/FormTagHelper/text_field_tag)

## **label\_tag**

[https://apidock.com/rails/ActionView/Helpers/FormTagHelper/label\\_tag](https://apidock.com/rails/ActionView/Helpers/FormTagHelper/label_tag)

# Melhorando o show (Coin e MiningType)

Veja o exemplo...

```
<form>

  <div class="form-group">
    <%= label_tag nil, @coin.class.human_attribute_name(:description) %>
    <%= text_field_tag nil, @coin.description, class:"form-control", disabled:
true %>
  </div>
  ...
</form>

<%= link_to 'Edit', edit_coin_path(@coin), class:"btn btn-primary" %> |
<%= link_to 'Back', coins_path, class:"btn btn-primary" %>
```

# Conhecendo o Heroku

# Conhecendo o Heroku

O Heroku é uma plataforma em nuvem como um serviço (**PaaS - Platform as a Service**) que suporta várias linguagens de programação.

`https://www.heroku.com/`

**Cadastre-se no site!**

# Conhecendo o Heroku

## Heroku CLI (Command Line Interface)

<https://devcenter.heroku.com/articles/heroku-cli>

### Ubuntu

```
>> cat /etc/issue  
>> sudo snap install --classic heroku  
>> heroku --version
```

# **Publicando nossa aplicação (Deploy)**



# Publicando nossa aplicação (Deploy)

Siga os passo a passo para aplicações Rails 5.x na documentação do Heroku...

`https://devcenter.heroku.com/articles/getting-started-with-rails5`

# Publicando nossa aplicação (Deploy)

## Ajustes...

```
gem 'webpacker'
```

```
rails webpacker:install
```

```
//= require jquery/dist/jquery
```

```
config.assets.js_compressor = Uglifier.new(harmony: true, compress: {  
  unused: false })
```