PMR3304 - Sistemas de Informação - 02

Aula 02

Prof. Dr. Marcos de Sales Guerra Tsuzuki

09 de Agosto de 2019

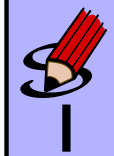
PMR-EPUSP

Gerenciando o Fluxo de Dados

Gerenciando o Fluxo de Dados - Controller

Controllers são objetos Ruby, e são armazenados no diretório **app/controller**. Cada controller possui um nome, e o objeto interno ao arquivo de controle é chamado de *nome*Controller. Para exemplificar o uso de controles, vamos criar uma aplicação que armazena o nome de pessoas que passaram por um site, e vamos chamá-lo de *guestbook*.

Execute o comando **rails new guestbook**. Avance para o o diretório recém criado **cd guestbook**, e em seguida crie um controle **rails generate controller entries**.



Para você fazer em casa

Execute os comandos acima e crie o controle.

O conteúdo padrão do arquivo **app/controller/entries_controller.rb**

```
class EntriesController < ApplicationController
end
```

Para que este controle realize algo, será acrescentado um método **sign_in**.

```
class EntriesController < ApplicationController
  def sign_in
  end
end
```

Vamos associar uma view para este controller. Um arquivo de nome **sign_in.html.erb** será criado no diretório **app/view/entries/** conforme a listagem abaixo:

```
<h1>Hello <%= @name %></h1>
<%= form_tag action: 'sign_in' do %>
  <p>Entre o seu nome:
  <%= text_field_tag 'visitor_name', @name %></p>
  <%= submit_tag 'Sign in' %>
<% end %>
```

- Neste código existem diversos métodos helpers. O método *form_tag* possui o nome do controller, como parâmetro do *:action*.

Gerenciando o Fluxo de Dados - Controller

```
<h1>Hello <%= @name %></h1>
<%= form_tag action: 'sign_in' do %>
  <p>Entre o seu nome:
  <%= text_field_tag 'visitor_name', @name %></p>
  <%= submit_tag 'Sign in' %>
<% end %>
```

- O método *text_field_tag* requer dois parâmetros e os utiliza para criar um campo do form. O primeiro, **visitor_name**, é o identificador que o form utilizará para descrever o campo e enviá-lo de volta para o controller, enquanto o segundo é o texto padrão que campo possui. Se o usuário já tiver preenchido este formulário alguma vez, a variável *@name* já estará povoada.

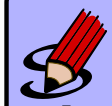
```
<h1>Hello <%= @name %></h1>
<%= form_tag action: 'sign_in' do %>
  <p>Entre o seu nome:
  <%= text_field_tag 'visitor_name', @name %></p>
  <%= submit_tag 'Sign in' %>
<% end %>
```

- O último método é *submit_tag* cria o botão que enviará os dados do form de volta para o controller quando ele for pressionado.

Será necessário habilitar o roteamento para o controller recém criado. Assim, o arquivo **config/routes.rb** será editado para incluir as seguintes linhas:

```
get 'entries/sign_in' => 'entries#sign_in'
post 'entries/sign_in' => 'entries#sign_in'
```

Inicie o servidor e visite a página `http://localhost:3000/entries/sign_in`.



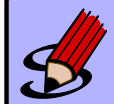
Para você fazer em casa

Confirme o funcionamento do controller e view recém criados.

Dando vida ao método *sign_in*:

```
class EntriesController < ApplicationController
  def sign_in
    @name = params[:visitor_name]
  end
end
```

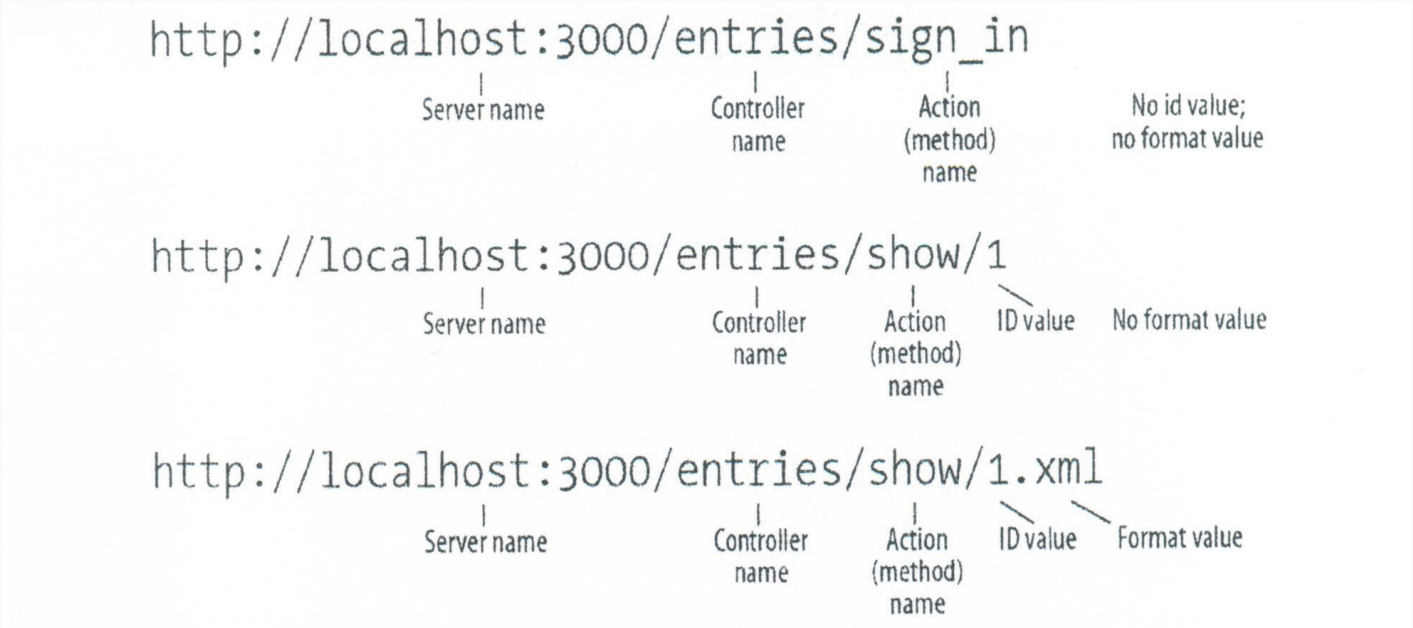
Inicie o servidor e visite a página `http://localhost:3000/entries/sign_in`. Ao executar a primeira vez, nenhum nome aparecerá. Mas, ao pressionar o botão, visitando pela segunda vez, as boas vindas ao nome aparecerá.



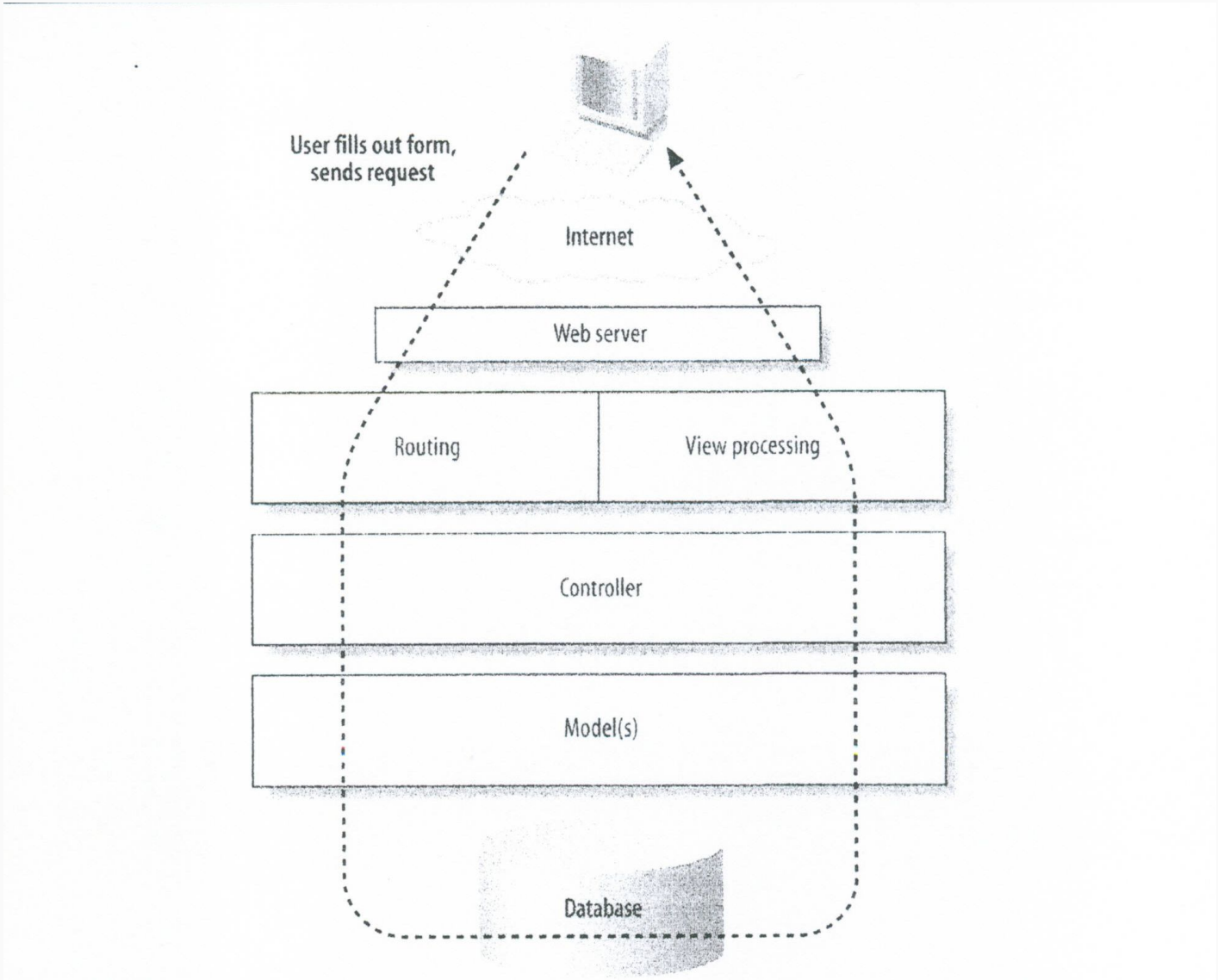
Para você fazer em casa

Confirme o funcionamento do controller e view recém criados.

Gerenciando o Fluxo de Dados - Controller



Gerenciando o Fluxo de Dados - Controller



Inicialmente verifique se o banco de dados está funcionando corretamente e tecle **sqlite3 -help** na linha de comando.

Vamos criar o primeiro model **rails generate model entry**. Serão criados dois arquivos importantes: **app/models/entry.rb** e **db/migrate/*_create_entries.rb**. O segundo arquivo contém:

```
class CreateEntries < ActiveRecord::Migration[5.2]
  def change
    create_table :entries do |t|


      t.timestamps
    end
  end
end
```

```
class CreateEntries < ActiveRecord::Migration[5.2]
  def change
    create_table :entries do |t|


      t.timestamps
    end
  end
end
```

Este código cria uma tabela para *entries* que é o plural para *entry*. Nos casos onde o plural e o singular coincidem, um “s” adicional é incluído para diferenciar os dois. *ActiveRecord* é a classe que gerencia a conexão com o banco de dados.

Acrescentando um campo à tabela. É importante saber que o Rails está criando um campo como chave primária **:id** para a tabela. O campo **t.timestamps** existe para gerenciamento do banco, assim saberemos o momento em que um campo foi inserido ou modificado.

```
class CreateEntries < ActiveRecord::Migration[5.2]
  def change
    create_table :entries do |t|
      t.string :name
      t.timestamps
    end
  end
end
```

```
class CreateEntries < ActiveRecord::Migration[5.2]
  def change
    create_table :entries do |t|
      t.string :name
      t.timestamps
    end
  end
end
```

Execute o migration para efetivar a modificação: **rails db:migrate**.

Vamos observar o arquivo **app/models/entry.rb**.

```
class Entry < ApplicationRecord
end
```

Observe que ele deriva da classe **ApplicationRecord** que está definida no mesmo diretório.

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Esta é uma classe *abstrata*, portanto ela não pode ser instanciada.

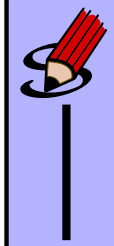
O controller é o ponto chave para a comunicação entre form e model. Vamos retornar ao arquivo **app/controllers/entry_controller.rb** e modificá-lo para

```
class EntriesController < ApplicationController
  def sign_in
    @name = params[:visitor_name]
    @entry = Entry.create({:name => @name})
  end
end
```

Esta nova linha de comando é a compactação de três comandos em um. O primeiro comando cria uma nova variável **@entry**, a linha seguinte configura a propriedade **name** e a terceira linha salva o valor na tabela.

```
@entry = Entry.new
@entry.name = @name
@entry.save
```

Execute o servidor e acesse novamente com o browser `http://localhost:3000/entries/sign_in`. Será possível identificar alguns comandos coloridos em SQL no prompt de comando. O banco de dados foi acessado (comando SELECT) para recuperar informação armazenada e o banco de dados foi atualizado (comando INSERT).



Para você fazer em casa

Confirme o funcionamento do controller e model recém criados.

Foram inseridos brancos no banco de dados. Para limparmos o banco de dados vamos reverter executando **rails db:rollback** e em seguida recriar o banco de dados com **rails db:migrate**.

Para você fazer em casa

Retornar o banco de dados para o estado vazio, conforme os comandos descritos.

Para evitar que brancos sejam inseridos indevidamente no banco de dados, o arquivo **app/controllers/entry_controller.rb** deve ser modificado.

```
class EntriesController < ApplicationController
  def sign_in
    @name = params[:visitor_name]
    unless @name.blank?
      @entry = Entry.create({:name => @name})
    end
  end
end
```

Para você fazer em casa

Confirme o funcionamento do controller e model.

Agora vamos recuperar as entradas existentes no banco de dados e exibi-las. O objeto **Entry** inclui um método de busca **Entry.all** (como **new** e **save**) herdado de **ActiveRecord::Base**. O arquivo **app/controllers/entry_controller.rb** deve ser modificado.

```
class EntriesController < ApplicationController
  def sign_in
    @name = params[:visitor_name]
    unless @name.blank?
      @entry = Entry.create({:name => @name})
    end
    @entries = Entry.all
  end
end
```

A view **app/view/entries/sign_in.html.erb** também deve ser modificada para exibir os registros recuperados.

```
<h1>Hello <%= @name %></h1>
  <%= form_tag action: 'sign_in' do %>
    <p>Entre o seu nome:
    <%= text_field_tag 'visitor_name', @name %></p>
    <%= submit_tag 'Sign in' %>
  <% end %>
  <p>Visitantes anteriores:</p>
  <ul>
  <% @entries.each do |entry| %>
    <li><%= entry.name %></li>
  <% end %>
  </ul>
```

A view **app/view/entries/sign_in.html.erb** também deve ser modificada para exibir os registros recuperados.

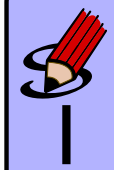
Para você fazer em casa

Confirme o funcionamento do controller e model recém criados.

Agora é possível analisar o banco de dados diretamente. Entre no console do Rails utilizando o comando **rails c**.

- O comando **Entry.find 1** recupera o registro número 1 da tabela **Entry**.
- O comando **Entry.all** recupera todos os registros da tabela **Entry**.
- O comando **Entry.first** recupera o primeiro registro da tabela **Entry**.
- O comando **Entry.last** recupera o último registro tabela **Entry**.
- O comando **Entry.where(name: "Marcos")** recupera o registro da tabela **Entry** com nome “Marcos”.
- O comando **Entry.order(:name)** recupera os registros da tabela **Entry** ordenados pelo nome.

Agora é possível analisar o banco de dados diretamente. Entre no console do Rails utilizando o comando **rails c**.



Para você fazer em casa

Confirme o funcionamento dos comandos explicados.

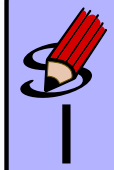
Scaffold e REST

Vamos desenvolver a mesma aplicação de modo mais completo utilizando novas ferramentas do Rails. Um ponto chave para este desenvolvimento é o **scaffold**. Assim, remova o diretório contendo a aplicação desenvolvida anteriormente e inicie uma nova com **rails new guestbook**. Em seguida avance para o diretório recém criado **cd guestbook**. Vamos agora criar um modelo com suporte para scaffold **rails generate scaffold Person name:string**. Este comando realizará uma série de subcomandos:

Scaffold e REST - Introdução

- Uma *data migration* para estabelecer as tabelas necessárias para o modelo;
- Um modelo (com acompanhamento de testes e reparos para os testes);
- Um nova rota que mapeará as solicitações do usuário para o controle;
- Um controller para enviar dados entre os diversos componentes;
- Quatro views (*index*, *edit*, *show* e *new*) e o suporte a um form que reduz a duplicação de códigos *_form.html.erb*;
- Testes para o controller;
- Um arquivo vazio para métodos helper;
- Um arquivo CoffeeScript e duas stylesheets (people e scaffold) para todas as views.

Para efetivar a mudança, vamos executar **rails db:migrate**, em seguida executamos o servidor **rails server**. Finalmente, execute no browser o comando `http://localhost:3000/people`



Para você fazer em casa

Confirme o funcionamento dos comandos explicados.

Vamos executar algumas simulações com o ambiente desenvolvido:

- Clique em *New Person*, e forneça um nome e clique em *Create Person*. O nome foi inserido ao cadastro.
- Clique em *Edit*, e agora é possível modificar o nome fornecido.
- Ao clicar em *Back*, o sistema retorna à página original com a lista de nomes.

REST é uma proposta que cria aplicações WEB obtendo vantagens da forma como o protocolo WEB foi definido.

- Os usuários verificarão que as aplicações WEB funcionarão como eles desejam, de acordo com os browswer. As páginas poderão ser marcadas para retorno futuro, caso as URLs tenham significado.
- Os administradores de rede poderão utilizar suas técnicas preferidas para gerenciar o tráfego WEB sem preocupação em bloquear aplicações.
- O desenvolvedor é que obtém os maiores benefícios, pois a arquitetura baseada em REST é muito próxima da proposta Rails MVC e torna mais simples entender o significado de cada trecho de código e o seu objetivo.

Tabela 1: Os principais métodos do protocolo HTTP e o cenário de utilização de cada um deles:

GET	Obter os dados de um recurso
POST	Criar um novo recurso
PUT	Substituir os dados de um determinado recurso
PATCH	Atualizar parcialmente um determinado recurso.
DELETE	Excluir um determinado recurso.
HEAD	Similar ao **GET**, mas utilizado apenas para se obter os cabeçalhos de resposta sem os dados em si.
OPTIONS	Obter quais manipulações podem ser realizadas em um determinado recurso.

Tabela 2: Veja a seguir o padrão de utilização dos métodos HTTP em um serviço REST, que é utilizado pela maioria das aplicações e pode ser considerado uma boa prática. Como exemplo será utilizado um recurso chamado Cliente.

Método	URI	Utilização
GET	/clientes	Recuperar os dados dos clientes.
GET	/clientes/id	Recuperar os dados de um cliente.
POST	/clientes	Criar um novo cliente.
PUT	/clientes/id	Atualizar os dados de um cliente.
DELETE	/clientes/id	Excluir um determinado cliente.

Os desenvolvedores WEB utilizam erroneamente dois métodos HTTP para obter informações dos sites: **GET** e **POST**. Com capacidade de 1024 caracteres, o **GET** é utilizado quando se quer passar poucas informações para realizar uma pesquisa através da URL. A função do **GET** é recuperar um recurso existente no servidor. O resultado de uma requisição **GET** é "cacheável" pelo cliente, ou seja, fica no histórico do navegador. O método **POST** utiliza a URI (de Uniform Resource Identifier) para envio de informações ao servidor. A URI não é retornável ao cliente, o que torna o método **POST** mais seguro, pois não expõe os dados enviados no navegador. Como não tem limite de capacidade para envio de informações, este método se torna melhor que o **GET**. No **POST**, uma conexão paralela é aberta e os dados são passados por ela.

Como boa prática, em relação aos métodos do protocolo HTTP, evite utilizar apenas o método **POST** nas requisições que alteram o estado no servidor, tais como: cadastro, alteração e exclusão, e principalmente, evite utilizar o método **GET** nesses tipos de operações, pois é comum os navegadores fazerem cache de requisições **GET**, as disparando antes mesmo do usuário clicar em botões e links em uma pagina HTML.

Analisando o controller **app/controllers/people_controller.rb** em detalhe. Como é um arquivo grande, vamos dividir em cinco partes. Abaixo, temos a primeira parte.

```ruby
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :destroy]
  private
    # Use callbacks to share common setup or constraints between
        actions.
    def set_person
      @person = Person.find(params[:id])
    end
    # Never trust parameters from the scary internet, only allow the
        white list through.
    def person_params
      params.require(:person).permit(:name)
    end
end
```

A ação **set_person** deve ser executada antes que o acionamento dos métodos **show**, **edit**, **update** e **destroy**.

```ruby
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :destroy]
  # GET /people # GET /people.json
  def index
    @people = Person.all
  end
  # GET /people/1 # GET /people/1.json
  def show
  end
  # GET /people/new
  def new
    @person = Person.new
  end
  # GET /people/1/edit
  def edit
  end
```

Gerenciando o Fluxo de Dados - Conectando o Controller ao Model

```
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :destroy]
  # POST /people # POST /people.json
  def create
    @person = Person.new(person_params)
    respond_to do |format|
      if @person.save
        format.html { redirect_to @person, notice: 'Person was successfully
            created.' }
        format.json { render :show, status: :created, location: @person }
      else
        format.html { render :new }
        format.json { render json: @person.errors, status: :
            unprocessable_entity }
      end
    end
  end
```

```
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :
      destroy]

  # PATCH/PUT /people/1
  # PATCH/PUT /people/1.json
  def update
    respond_to do |format|
      if @person.update(person_params)
        format.html { redirect_to @person, notice: 'Person was
            successfully updated.' }
        format.json { render :show, status: :ok, location: @person
             }
      else
        format.html { render :edit }
        format.json { render json: @person.errors, status: :
            unprocessable_entity }
      end
    end
  end
```

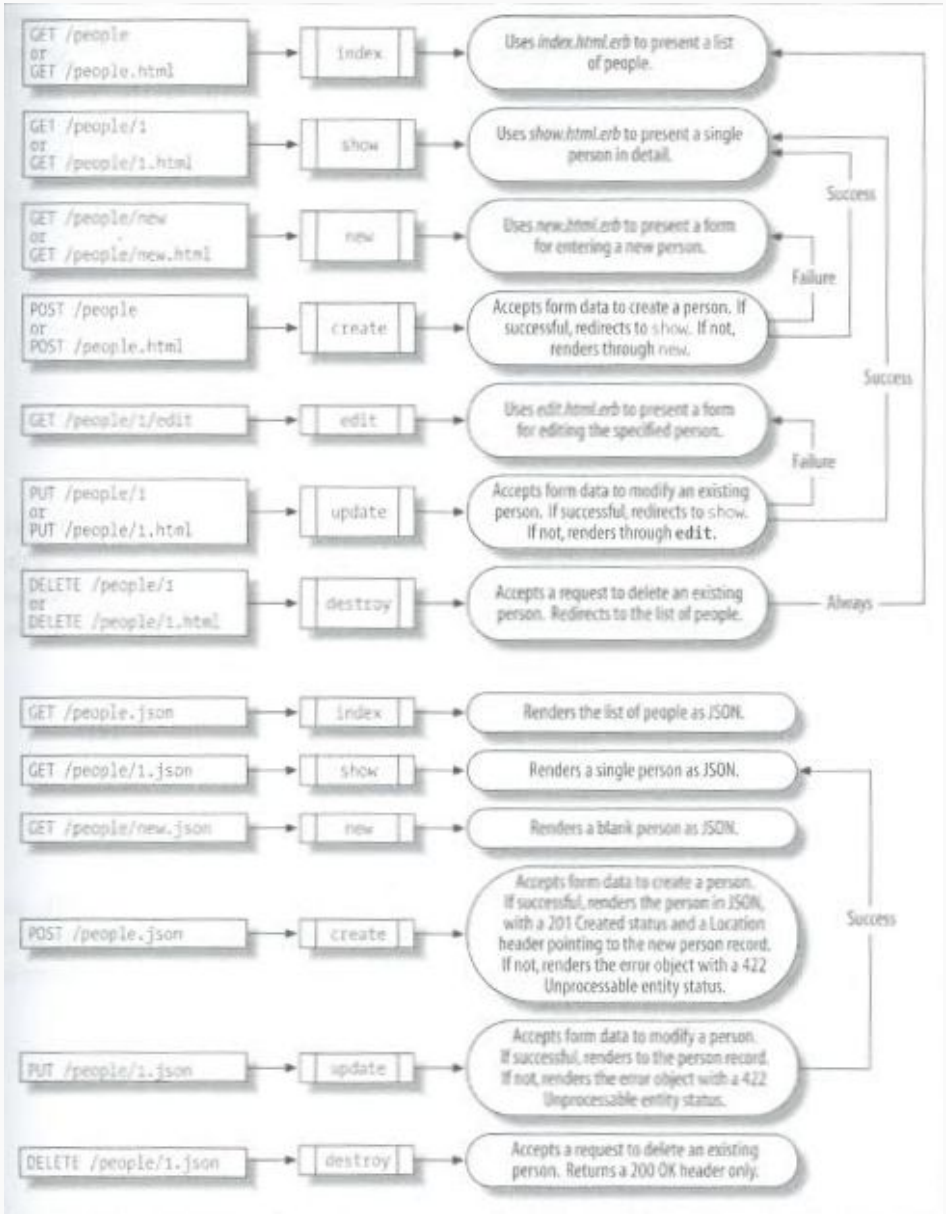
```
class PeopleController < ApplicationController
  before_action :set_person, only: [:show, :edit, :update, :destroy]

  # DELETE /people/1
  # DELETE /people/1.json
  def destroy
    @person.destroy
    respond_to do |format|
      format.html { redirect_to people_url, notice: 'Person was successfully
          destroyed.' }
      format.json { head :no_content }
    end
  end
end
```

Tabela 3: O conjunto completo de rotas utilizado em Rails.

Prefixo	Método	URI	Utilização
people	**GET**	/people(.:format)	people#index
	POST	/people(.:format)	people#create
new_person	**GET**	/people/new(.:format)	people#new
edit_person	**GET**	/people/:id/edit(.:format)	people#edit
person	**GET**	/people/:id/edit(.:format)	people#show
	PATCH	/people/:id/edit(.:format)	people#update
	PUT	/people/:id/edit(.:format)	people#update
	DELETE	/people/:id/edit(.:format)	people#destroy

Gerenciando o Fluxo de Dados - Controller



The End!