

**INSTITUTO FEDERAL**

Paraná

Campus Paranaguá

# Ruby on Rails

API REST JSON

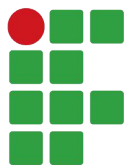
Desenvolvimento Web III

Prof. Diego Stiehl

# Views e HTML

---

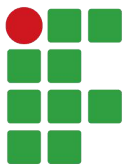
- O Rails é um framework de back-end
- Todo o processamento acontece no servidor
- Como resposta temos, normalmente, HTML
  - Devidamente customizado de acordo com os parâmetros da requisição
  - Coordenada pelas **views**
- O browser recebe este HTML estático
  - E renderiza na janela



# Outras Respostas

---

- Nem sempre nossa aplicação vai gerar apenas HTML como resposta
  - Sem interesse direto do browser
- As informações do nosso servidor podem ser solicitadas para outros fins
  - AJAX
  - Outras aplicações web
  - Frameworks de front-end
    - React, Vue, Angular, ...
  - Aplicações mobile



# JSON

---

- O que retornar então?
  - JSON
- JSON se tornou o padrão nos últimos anos
- Qualquer linguagem tem bibliotecas para “parsear” dados em JSON

# Gerar JSON

---

- Para fazer uma action retornar JSON basta utilizarmos o método `render`

```
def index
  @articles = Kind.all
  render json: @kinds
end
```

- Objetos já sabem se converter para JSON
  - Estrutura padrão



# Retorno

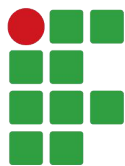
- Acesse <http://localhost:3000/kinds>

```
[
  {
    "id": 1,
    "description": "Alternative Dispute Resolution",
    "created_at": "2019-10-17T12:42:59.094Z",
    "updated_at": "2019-10-17T12:42:59.094Z"
  },
  {
    "id": 2,
    "description": "Package / Freight Delivery",
    "created_at": "2019-10-17T12:42:59.105Z",
    "updated_at": "2019-10-17T12:42:59.105Z"
  }
]
```

# API / Endpoint

---

- Aplicações web que exportam JSON são chamadas de APIs
  - Porque retornam parte do dado que outras aplicações precisam para executar
- As URLs em APIs são chamadas de endpoints
  - <http://localhost:3000/kinds>
  - <http://localhost:3000/contacts/1>

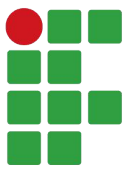


# JSend

---

- É interessante botar alguma ordem nos dados trafegados via JSON
  - Outras pessoas podem consumir os dados
- JSend é um padrão beeeem simples
  - E bem difundido
- <https://github.com/omniti-labs/jsend>





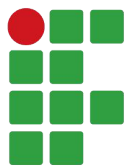
INSTITUTO  
FEDERAL

# JSend

```
{
  "status": "success",
  "data": {
    "kind": {
      "id": 1,
      "description": "Alternative Dispute Resolution"
    }
  }
}
```

```
{
  "status": "fail",
  "data": {
    "id": "No ID were provided"
  }
}
```

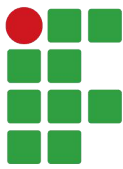
```
{
  "status": "error",
  "message": "Internal server error"
}
```



# Prática

---

- KindsController
  - Altere as ações index e show
  - Faça elas retornarem JSON usando JSend
- Observação
  - Para a prática, ignore a possibilidade de retornar HTML



# Problema

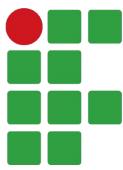
- Podemos retornar qualquer coisa com o método render
- Porém teremos problema conforme a complexidade dos objetos aumenta

```
render json: { ready: 'ok' }
```

↑  
Susse

Vish! →

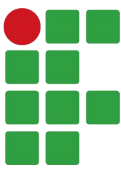
```
render json: { ready: "ok",  
message: "#{kind.description}  
successfully registered.",  
values: [{ accepted: true,  
size: 30 }, { accepted:  
false, size: 15 }] }
```



# Jbuilder

---

- Jbuilder é a solução para o problema da complexidade dos objetos
- Com ele, os JSON são considerados views
- No lugar de renderizar uma view HTML, iremos renderizar uma view JSON
- Funciona de maneira semelhante ao ERB
- Escrevemos código que parece o JSON final



# Como Usar

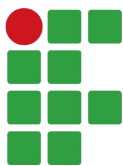
---

- Criar arquivos nas pastas de views
  - `index.json.builder`
  - `show.json.builder`
- Se o requisitante estiver esperando JSON como resposta:
  - O Rails envia a view em JSON ao invés do ERB
- Convenção sobre Configuração

# Esperando JSON?

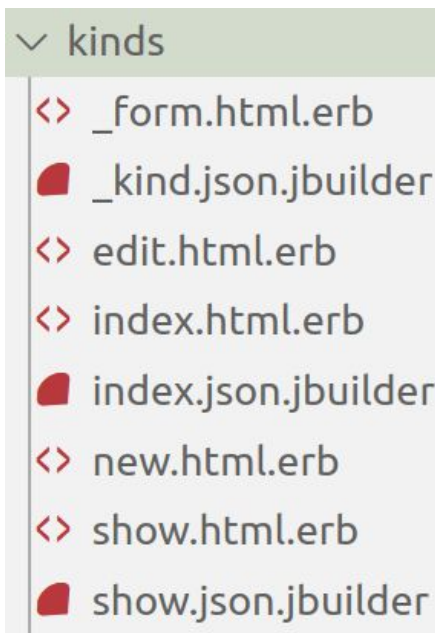
---

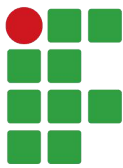
- Como sei que o cliente espera JSON?
- Alguma destas possibilidades
  - Terminação da URL
    - <http://localhost:3000/kinds/1.json>
  - Cabeçalho HTTP na requisição
    - Accept: application/json
  - Se meu endpoint sabe somente JSON
    - É o caso de APIs
      - Não há versão HTML do endpoint



# Scaffold

- Os controllers/views gerados por scaffold já utilizam **Jbuilder** por padrão
- Dê uma olhada na pasta views





# Estrutura Jbuilder

- Cada view Jbuilder tem um objeto **json**
  - As propriedades que definirmos nele serão exportadas para o JSON
- Exemplo:

`json.kinds @kinds`

Objeto vindo do Controller


Propriedade a ser criada no JSON

```
{
  "kinds": [
    {
      "id": 1,
      "description": "Alternative Dispute Resolution",
      "created_at": "2019-10-17T12:42:59.094Z",
      "updated_at": "2019-10-17T12:42:59.094Z"
    },
    {
      "id": 2,
      "description": "Package / Freight Delivery",
      "created_at": "2019-10-17T12:42:59.105Z",
      "updated_at": "2019-10-17T12:42:59.105Z"
    }
  ]
}
```

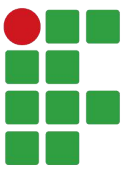


# Imprimir Array

json.array! @kinds



```
[
  {
    "id": 1,
    "description": "Alternative Dispute Resolution",
    "created_at": "2019-10-17T12:42:59.094Z",
    "updated_at": "2019-10-17T12:42:59.094Z"
  },
  {
    "id": 2,
    "description": "Package / Freight Delivery",
    "created_at": "2019-10-17T12:42:59.105Z",
    "updated_at": "2019-10-17T12:42:59.105Z"
  }
]
```



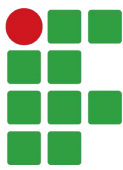
# Iterar Array

- Podemos usar um bloco para detalhar cada item de um array

```
json.array! @kinds do |kind|  
  json.id kind.id  
  json.description kind.description  
end
```

Propriedades a serem  
criadas no JSON

```
[  
  {  
    "id": 1,  
    "description": "Alternative Dispute Resolution"  
  },  
  {  
    "id": 2,  
    "description": "Package / Freight Delivery"  
  }  
]
```



# Extract

- Se o que precisarmos for apenas algumas propriedades específicas do model:

```
json.array! @kinds do |kind|  
  json.extract! kind, :id, :description  
end
```

Propriedades a serem  
criadas no JSON

```
[  
  {  
    "id": 1,  
    "description": "Alternative Dispute Resolution"  
  },  
  {  
    "id": 2,  
    "description": "Package / Freight Delivery"  
  }  
]
```

# Criar Propriedades

- Podemos “inventar” propriedades

```
json.array! @kinds do |kind|  
  json.full_name kind.description  
  json.fake "I'm not in the model (#{kind.id})"  
end
```

Propriedades a serem  
criadas no JSON

```
[  
  {  
    "full_name": "Alternative Dispute Resolution",  
    "fake": "I'm not in the model (1)"  
  },  
  {  
    "full_name": "Package / Freight Delivery",  
    "fake": "I'm not in the model (2)"  
  }  
]
```

# Objetos Aninhados

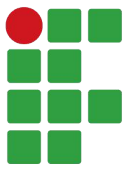
```
json.array! @kinds do |kind|  
  json.extract! kind, :id, :description  
  json.contacts kind.contacts do |contact|  
    json.name contact.name  
    json.phone_count contact.phones.count  
  end  
end
```

```
[  
  {  
    "id": 1,  
    "description": "Alternative Dispute Resolution",  
    "contacts": [  
      {  
        "name": "Yoda",  
        "phone_count": 3  
      },  
      {  
        "name": "Lyra Erso",  
        "phone_count": 3  
      }  
    ]  
  },  
  {  
    "id": 2,  
    "description": "Package / Freight Delivery",  
    "contacts": [  
      {  
        "name": "Darth Vader",  
        "phone_count": 3  
      }  
    ]  
  }  
]
```

# Partials

---

- Podemos alocar trechos do código em outros arquivos do Jbuilder
  - Evita o aninhamento excessivo
  - Possibilita reaproveitamento de código
  - Organiza melhor as responsabilidades
- Arquivos são chamados de partials
  - Servem só para serem incluídos em outros
- Nome do arquivo com underline
  - \_kind.json.jbuilder



# Partials

kinds/index.json.jbuilder

```
json.array! @kinds, partial: "kinds/kind", as: :kind
```

kinds/\_kind.json.jbuilder

```
json.extract! kind, :id, :description
```

```
json.contacts kind.contacts, partial: "contacts/contact", as: :contact
```

contacts/\_contact.json.jbuilder

```
json.name contact.name
```

```
json.phone_count contact.phones.count
```

# Ação show

- A partial criada permite reaproveitar para a ação de show

kinds/show.json.builder

```
json.partial! "kinds/kind", kind: @kind
```

```
{
  "id": 1,
  "description": "Alternative Dispute Resolution",
  "contacts": [
    {
      "name": "Yoda",
      "phone_count": 3
    },
    {
      "name": "Lyra Erso",
      "phone_count": 3
    }
  ]
}
```



# Prática

---

- Formate o JSON das rotas de index e show para utilizar JSend
- Lembre que um ID inválido gera uma falha

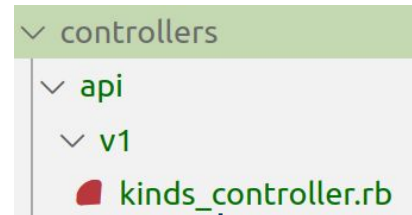
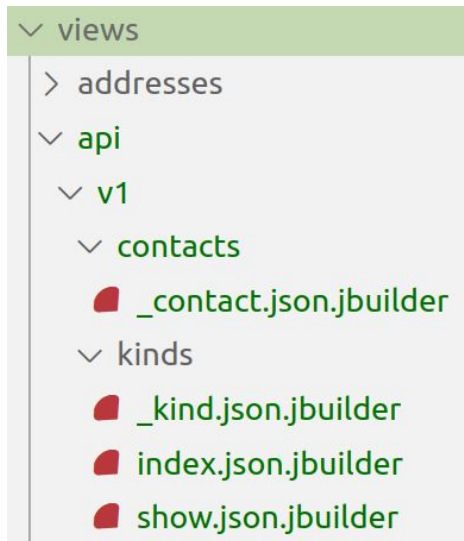
# Rotas de API

- Às vezes iremos criar APIs mais complexas
  - Com muitos endpoints
- Neste caso é melhor separarmos todas as rotas de API das rotas “web”
  - <http://server/kinds> ← Rota web (para HTML)
  - <http://server/api/v1/kinds> ← Rota da API (para JSON)

↑  
Versionamento da API  
Para evitar parâmetros e respostas inesperados por usuários externos da API

# Rotas de API

- Para funcionar, precisamos usar namespaces



```
class Api::V1::KindsController < ApplicationController
  # Actions do Controller
end
```

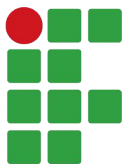
```
routes.rb
Rails.application.routes.draw do
  # ... Outras rotas
  namespace :api do
    namespace :v1 do
      resources :kinds
    end
  end
end
```

# Envio de Dados

- Nossos formulários Rails tem uma forma própria de organizar as informações

```
▼ Form Data    view source    view URL encoded  
  
authenticity_token: W9ISmtQKI4ToheRfgQvLnpMUGsUFysz0tRns+dtZh1IPxVcUqUtPPZ1mq9  
7DhCFJJ1PJF87irEZnx2Rfp+4C5w==  
kind[description]: Novo tipo de contato  
commit: Criar Kind
```

- Até podemos utilizar este formato
  - Mas clientes de APIs preferem enviar JSON puro no corpo da requisição



INSTITUTO  
FEDERAL

# Enviando JSON

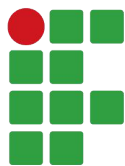
The screenshot displays a REST client interface with the following components:

- Request Section:**
  - Method: **POST**
  - URL: `http://localhost:3000/kinds`
  - Buttons: **Send** and **Save**
- Request Body:**
  - Tab: **Body** (selected)
  - Format: **raw** (selected)
  - Content: 

```
1 {
2   "kind": {
3     "description": "Alo"
4   }
5 }
```
- Response Section:**
  - Tab: **Body** (selected)
  - Format: **JSON** (selected)
  - Content: 

```
1 {
2   "id": 7,
3   "description": "Alo",
4   "contacts": []
5 }
```
- Metadata:**
  - Status: **201 Created**
  - Time: **113ms**
  - Size: **1.04 KB**
  - Buttons: **Save Response**

An arrow points from the text "Action show renderizada após inclusão" to the response body, indicating that the action is rendered after the inclusion.



# Desafio

---

- Telefones precisam ser confirmados
- Ao lado de cada telefone, coloque:
  - O status (confirmado / não)
  - Um botão
- Botão dispara requisição AJAX para:
  - `/api/v1/contacts/ID/phones/ID/confirm`
- Ao voltar resposta:
  - Atualizar status

# Enviando JSON

- Ajustar leitura de parâmetros

```
class Api::V1::KindsController < ApplicationController
  skip_before_action :verify_authenticity_token

  def create
    @kind = Kind.new(kind_params)
    if @kind.save
      render :show, status: :created, location: @kind
    else
      render json: @kind.errors, status: :unprocessable_entity
    end
  end

  private
  def kind_params
    json_params = ActionController::Parameters.new(JSON.parse(request.body.read))
    json_params.require(:kind).permit(:description)
  end
end
```