



Kurs programowania Ruby on Rails

Tworzenie Własnego API

Problemy z komunikacją



- Udostępniamy nasze dane do innych aplikacji
- Pobieramy dane z innych aplikacji i później je wykorzystujemy np. aktualna pogoda
- Dostarczamy dane do innych aplikacji np. porównywarka cen
- Potrzebujemy z pdfa odczytać kody obsługi błędów i wprowadzić je do programu

POTRZEBNY NAM FORMAT WYMIANY DANYCH!

XML



- XML, czyli eXtensible Markup Language (rozszerzalny język znaczników)
- jest formatem wymiany danych, w założeniu bardzo prosty
- może być stosowany jako plik konfiguracyjny
- wspierany bardzo przez Microsoft w pewnym momencie
- stworzony w 1996 roku
- używany np. przez porównywarki produktów
- specyfikacja języka <http://pl.wikipedia.org/wiki/XML>
- wspierany przez większość API

Przykładowy plik XML



```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<ksiazka-telefoniczna kategoria="bohaterowie książek">
```

```
<!-- komentarz -->
```

```
<osoba charakter="dobry">
```

```
<imie>Ambroży</imie>
```

```
<nazwisko>Kleks</nazwisko>
```

```
<telefon>123-456-789</telefon>
```

```
</osoba>
```

Przykładowy plik XML c.d.



```
<osoba charakter="zły">
```

```
  <imie>Alojzy</imie>
```

```
  <nazwisko>Bąbel</nazwisko>
```

```
  <telefon/>
```

```
</osoba>
```

```
</ksiazka-telefoniczna>
```

API



- Application Programming Interface (API) sposób rozumiany jako ściśle określony zestaw reguł i opisów, w jaki programy komunikują się pomiędzy sobą.
- Zadaniem API jest dostarczenie odpowiednich specyfikacji i wymaganych protokołów komunikacyjnych.

Przykładowy plik Ceneo specyfikacja



<https://panel.ceneo.pl/documents/Informacje%20na%20temat%20struktury%20pliku%20xml.pdf>

Ruby on Rails XML



```
customer.projects.to_xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<projects type="array">
```

```
  <project>
```

```
    <amount type="decimal">20000.0</amount>
```

```
    <customer-id type="integer">1567</customer-id>
```

```
    <deal-date type="date">2008-04-09</deal-date>
```

Ruby on Rails XML c.d.



...

</project>

<project>

<amount type="decimal">57230.0</amount>

<customer-id type="integer">1567</customer-id>

<deal-date type="date">2008-04-15</deal-date>

...

</project>

</projects>

Ruby on Rails XML



```
[{:foo => 1, :bar => 2}, {:baz => 3}].to_xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<records type="array">
```

```
  <record>
```

```
    <bar type="integer">2</bar>
```

```
    <foo type="integer">1</foo>
```

```
  </record>
```

Ruby on Rails XML c.d.



```
<record>
```

```
  <baz type="integer">3</baz>
```

```
</record>
```

```
</records>
```

Czy może być coś prostszego niż XML?



- TAK JSON !!!!

JSON



- wspierany przez większość języków programowania
- w sposób naturalny wykorzystywany w Javascript i Ruby
- bardzo popularny obecnie , gdyż korzysta się z niego łatwiej niż XML, przyjemny w programowaniu

JSON przykład



```
{"menu": {  
  "id": "file",  
  "value": "File",  
  "popup": {  
    "menuitem": [  
      {"value": "New", "onclick": "CreateNewDoc()"},  
      {"value": "Open", "onclick": "OpenDoc()"},  
      {"value": "Close", "onclick": "CloseDoc()"}  
    ]  
  }  
}}
```

RAILS JSON



```
konata = User.find(1)
```

```
ActiveRecord::Base.include_root_in_json = true
```

```
konata.to_json
```

```
# => { "user": {"id": 1, "name": "Konata Izumi", "age": 16,  
           "created_at": "2006/08/01", "awesome": true} }
```

```
ActiveRecord::Base.include_root_in_json = false
```


RAILS JSON c.d.



```
konata.to_json
```

```
# => {"id": 1, "name": "Konata Izumi", "age": 16,  
      "created_at": "2006/08/01", "awesome": true}
```

Format



```
MacBook-Pro-Micha:respond_to michalos$ rake routes
```

Prefix	Verb	URI Pattern	Controller#Action
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(.:format)	people#show
	PATCH	/people/:id(.:format)	people#update
	PUT	/people/:id(.:format)	people#update
	DELETE	/people/:id(.:format)	people#destroy

Inne formaty przy generowaniu scaffolda



```
[{"name": "ads", "surname": "ads", "url": "http://localhost:3000/people/1.json"}]
```

Format



- Standardowo jeśli nie podamy formatu to generowany jest format html
- jeśli chcemy to zmienić to zmieniamy to w routes np:

```
resources :people, :defaults => { :format => 'json' }
```

- jeśli mamy format html to Railsy „szukają” plików z rozszerzeniem html, html.erb

Wspieranie jednego formatu przykład



```
def index
```

```
  @people = Person.all
```

```
  render :json => @people.to_json
```

```
end
```

Krócej...



```
def index
```


```
  @people = Person.all
```

```
  render :json => @people
```

```
end
```

Ćwiczenie 5 – przykładowa odpowiedź



← → ↻  localhost:3000/beers?page=3

```
[{"id":121,"name":"Sandra","brewery_name":"Kirk","created_at":"2013-11-13T15:45:55.922Z","updated_at":"2013-11-13T15:45:55.922Z"},
{"id":122,"name":"Alison","brewery_name":"Davies","created_at":"2013-11-13T15:45:55.925Z","updated_at":"2013-11-13T15:45:55.925Z"},
{"id":123,"name":"Barbara","brewery_name":"Taylor","created_at":"2013-11-13T15:45:55.927Z","updated_at":"2013-11-13T15:45:55.927Z"},
{"id":124,"name":"Kenneth","brewery_name":"Lee","created_at":"2013-11-13T15:45:55.931Z","updated_at":"2013-11-13T15:45:55.931Z"},
{"id":125,"name":"Sandra","brewery_name":"Smythe","created_at":"2013-11-13T15:45:55.934Z","updated_at":"2013-11-13T15:45:55.934Z"},
{"id":126,"name":"Roger","brewery_name":"Washington","created_at":"2013-11-13T15:45:55.937Z","updated_at":"2013-11-13T15:45:55.937Z"},
{"id":127,"name":"Michelle","brewery_name":"Kirby","created_at":"2013-11-13T15:45:55.941Z","updated_at":"2013-11-13T15:45:55.941Z"},
{"id":128,"name":"Cheryl","brewery_name":"Thatcher","created_at":"2013-11-13T15:45:55.943Z","updated_at":"2013-11-13T15:45:55.943Z"},
{"id":129,"name":"Simon","brewery_name":"Thompson","created_at":"2013-11-13T15:45:55.946Z","updated_at":"2013-11-13T15:45:55.946Z"},
{"id":130,"name":"Margaret","brewery_name":"Jackson","created_at":"2013-11-13T15:45:55.948Z","updated_at":"2013-11-13T15:45:55.948Z"}]
```

respond_to

- mówimy w akcji jakie są dopuszczalne formaty
- w bloku kodu od razu możemy renderować dany widok

```
def index
```

```
  @people = Person.all
```

```
  respond_to do |format|
```

```
    format.json { render json: @people }
```

```
    format.html
```

```
  end
```

```
end
```


JBUILDER



- gem używany standardowo przez RAILS do renderowania jsona.
- ja osobiście z niego nie korzystam
- generowany przy scaffoldzie
- <https://github.com/rails/jbuilder>
- rozszerzenie json.jbuilder

Przykłady JBUILDER 1



```
# @people = People.all
```

```
json.array! @people do |person|
```

```
  json.name person.name
```

```
  json.age calculate_age(person.birthday)
```

```
end
```

```
# => [ { "name": "David", "age": 32 }, { "name": "Jamie", "age":  
31 } ]
```

Przykłady JBUILDER 2



```
# @people = People.all
```

```
json.array! @people, :id, :name
```

```
# => [ { "id": 1, "name": "David" }, { "id": 2, "name": "Jamie" } ]
```

Przykłady JBUILDER 3



```
json.extract! @post, :id, :title, :content, :published_at
```

```
json.author do
```

```
  if @post.anonymous?
```

```
    json.null! # or json.nil!
```

```
  else
```

```
    json.first_name @post.author_first_name
```

```
    json.last_name @post.author_last_name
```

```
  end
```

```
end
```

Przykłady JBuilder 4



```
json.key_format! camelize: :lower
```

```
json.first_name 'David'
```

```
# => { "firstName": "David" }
```

Plik buildera tworzony jest w formacie nazwa_akcji.json.jbuilder

Inny Gem do Renderowania jsona



<https://github.com/nesquena/rabl>

DSL



Język dziedzinowy, także język dedykowany, język specjalizowany (ang. domain-specific language, DSL) to język programowania przystosowany do rozwiązywania określonej dziedziny problemów, określonej reprezentacji problemu lub określonej techniki ich rozwiązywania. Przeciwnieństwem języków dziedzinowych są języki programowania ogólnego zastosowania. Jbuilder i Rabl tworzą taki DSL

namespace



```
namespace :api do
```

```
  resources :posts
```

```
end
```

- mamy nowe routes z api/ścieżki

natomiast posts musi być w folderze api i mieć nazwę Api::Posts

Generowanie namespace api/posts



- tworzymy folder api w controllers
- tworzymy folder api w views
- zmiana ścieżek w widok
- generujemy scaffold Post name:string description:text
- przekopiuujemy wygenerowane widoki i kontrolery do odpowiednich folderów api

Generowanie namespace api/posts c.d.



Dopisujemy do kontrolera:

```
Api::PostsController
```

Dodajemy do routes:

```
namespace :api do
```

```
  resources :posts
```

```
end
```

Scope i path



scope '/admin' do

resources :posts, :comments

end

resources :posts, path: '/admin/posts'

W tych zastosowaniach mamy ścieżkę admin/posts, jednak kontroler nazywa się normalnie PostsController.rb

Problemy z API



- bezpieczeństwo (<http://railscasts.com/episodes/352-securing-an-api>)
- wersjonowanie



Kurs programowania Ruby on Rails

Tworzenie Własnego API

Koniec