

EPI: Interfejs Graficzny 2011/2012

Laboratorium nr 3

Podstawy Ruby on Rails

Aleksander Pohl

17 listopada 2011

Plan prezentacji

Biblioteka

Tworzenie nowego projektu

1. przejdź do katalogu, w którym ma znaleźć się główny katalog projektu
2. `▷ rails new library1`
3. `▷ cd library`
4. na końcu pliku Gemfile dodajemy wpis
`gem 'therubyracer'`
5. `▷ bundle`
6. `▷ rails server -p 3300`
 - ▶ domyślny numer portu to 3000; jeśli wiele aplikacji działa w osobnych serwerach, to możemy zmienić go za pomocą opcji `-p`
 - ▶ to polecenie wywołuje wbudowany serwer zwany WEBRICK
 - ▶ aby uruchomić w tle: `rails server -d`
7. otwórz przeglądarkę i wprowadź adres
<http://localhost:3300>

¹▷ oznacz znak zachetyv – nie wprowadzamy go!

Rails – okno powitalne



Welcome aboard

You're riding Ruby on Rails!

[About your application's environment](#)

Getting started

Here's how to get rolling:

1. **Use `script/generate` to create your models and controllers**

To see all available options, run it without parameters.

2. **Set up a default route and remove or rename this file**

Routes are set up in `config/routes.rb`.

3. **Create your database**

Run `rake db:migrate` to create your database. If you're not using SQLite (the default), edit `config/database.yml` with your username and password.

Rails – struktura katalogów

- ▶ app: kod źródłowy
- ▶ config: konfiguracja
- ▶ db: schemat bazy danych
- ▶ doc: dokumentacja
- ▶ lib: dodatkowe biblioteki
- ▶ log: logi
- ▶ public: treści statyczne
- ▶ script: specjalne skrypty aplikacji
- ▶ test: automatyczne testy
- ▶ tmp: pliki tymczasowe
- ▶ vendor: dodatki (pluginy)

Katalog 'app'

Railsy wykorzystują wzorzec projektowy Model-View-Controller i jest on odzwierciedlony w strukturze katalogów:

- ▶ **models** – klasy zawierające logikę biznesową
- ▶ **controllers** – kontrolery spajające widoki z danymi
- ▶ **views** – widoki zorganizowane w katalogach odpowiadających kontrolerom
- ▶ **helpers** – metody pomocnicze wykorzystywane w widokach
- ▶ **mailers** – dodatkowe moduły pomocne w wysyłaniu i odbieraniu poczty elektronicznej
- ▶ **assets** – zasoby: obrazki, javascript, css

Konfiguracja bazy danych (opcjonalne)

Parametry połączenia z bazą danych określone są w pliku **config/database.yml**:

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

Aby utworzyć bazę danych wywołujemy polecenie:

▷ rake db:create

Aby usunąć istniejącą bazę danych:

▷ rake db:drop

Tworzenie rusztowania dla modelu Author

Przechodzimy do katalogu libarary i generujemy rusztowanie:

```
▷ rails generate scaffold author first_name:string ↔  
last_name:string2
```

- ▶ model Author posiada atrybuty imię i nazwisko, które są łańcuchami znaków
- ▶ zostanie wygenerowana migracja, która dodaje do bazy danych schemat odpowiadający modelowi Author
- ▶ zostanie również wygenerowany podstawowy kontroler wraz z widokami do manipulowania autorami

²↔ oznacza kontynuację w tej samej linii! Tekst nie mieści się na slajdzie, dlatego jest w następnej linii.

Rusztowanie dla modelu Author

Generator rusztowania utworzył m.in. następujące pliki:

- ▶ **db/migrate/xxx_create_authors.rb** – definiuje tabelę authors
- ▶ **app/models/author.rb** – dynamicznie mapuje autora do tabeli w bazie danych³
- ▶ **app/controllers/authors_controller.rb** – definiuje akcje index, show, new, create, edit, update oraz destroy
- ▶ **app/views/authors/*** – widoki w plikach html.erb

³**Uwaga:** nazwa modelu jest w liczbie pojedynczej a tabeli – w liczbie mnogiej.

Podgląd pliku migracji i modelu

Otwieramy plik `db/migrate/xxx_create_authors.rb`.

```
class CreateAuthors < ActiveRecord::Migration
  def change
    create_table :authors do |t|
      t.string :first_name
      t.string :last_name

      t.timestamps
    end
  end
end
```

Otwieramy plik `app/models/author.rb`.

```
class Author < ActiveRecord::Base
end
```

Uruchamiamy migrację

```
▷ rake db:migrate
== CreateAuthors: migrating =====
-- create_table(:authors)
   -> 0.0041s
== CreateAuthors: migrated (0.0042s) ==
```

- ▶ rake jest narzędziem podobnym do GNU Make, który pozwala na uruchamianie zadań, takich jak np. wywołanie migracji bazy danych
- ▶ aby polecenie wykonało się poprawnie trzeba być w katalogu library!
- ▶ polecenie powoduje zmianę schematu bazy danych zgodnie z wygenerowaną wcześniej migracją

Wygenerowany schemat bazy

Jeśli nie było problemów z biblioteką bazy danych sqlite3 możemy zobaczyć wygenerowane tabele:

```
▷ sqlite3 db/development.sqlite3
```

```
SQLite version 3.6.2
```

```
Enter ".help" for instructions
```

```
Enter SQL statements terminated with a ";"
```

```
sqlite> .schema
```

```
SHOW CREATE TABLE authors;
```

```
CREATE TABLE 'authors' ('id' INTEGER PRIMARY KEY  
  AUTOINCREMENT NOT NULL, 'first_name' varchar(255),  
  'last_name' varchar(255));
```

```
SHOW CREATE TABLE schema_migrations;
```

```
CREATE TABLE 'schema_migrations' ('version'  
  varchar(255) NOT NULL);
```

Podgląd pliku rusztowania

Otwieramy plik `app/controllers/authors_controller.rb`.

```
class AuthorsController < ApplicationController
  # GET /authors
  # GET /authors.json
  def index
    @authors = Author.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => @authors }
    end
  end

  # GET /authors/1
  # GET /authors/1.json
  def show
    @author = Author.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render :json => @author }
    end
  end
  #...
end
```

Podgląd widoków i efekt końcowy

Otwieramy plik `app/views/show.html.erb`.

```
<p id="notice"><%= notice %></p>
<p>
  <b>First name:</b>
  <%= @author.first_name %>
</p>
<p>
  <b>Last name:</b>
  <%= @author.last_name %>
</p>
<%= link_to 'Edit', edit_author_path(@author) %> |
<%= link_to 'Back', authors_path %>
```

Uruchamiamy serwer (`rails server`) i otwieramy przeglądarkę na adresie `http://localhost:3300/authors`

Tworzenie rusztowania dla modelu Book

```
▷ rails generate scaffold book title:string ↵  
author:references
```

Następnie (w katalogu głównym!)

```
▷ rake db:migrate
```

Oglądamy wynik pod adresem (pamiętając o serwerze):

<http://localhost:3300/books>

Próba utworzenia nowej książki zakończy się niestety porażką!

Zmiana sposobu tworzenia książek

Dodajemy poniższy kod na początku pliku
`app/controllers/books_controller.rb`:

```
class BooksController < ApplicationController
  before_filter :find_authors, :only => [:new, :edit, :update, :create]

  # GET /books
  # ...
```

Oraz na jego końcu (przed słowem kluczowym `end`):

```
protected
  def find_authors
    @authors = Author.find(:all).map do |author|
      [author.first_name + ' ' + author.last_name, author.id]
    end
  end
end
```

Zmiana sposobu tworzenia książek – cd.(2)

Modyfikujemy plik `app/views/books/_form.html.erb`.

Zastępujemy:

```
<div class="field">
  <%= f.label :author %><br />
  <%= f.text_field :author %>
</div>
```

przez:

```
<div class="field">
  <%= f.label :author_id %><br />
  <%= f.select :author_id, @authors %>
</div>
```

Teraz zaglądamy pod `http://localhost:3300/books/new`

Wyświetlanie szczegółów książki

Zmieniamy widok książki `app/views/books/show.html.erb`:

```
<p>
  <b>Title:</b>
  <%= @book.title %>
</p>

<p>
  <b>Author:</b>
  <%= @book.author.first_name + ' ' +
    @book.author.last_name %>
</p>

<%= link_to 'Edit', edit_book_path(@book) %> |
<%= link_to 'Back', books_path %>
```

Teraz oglądamy szczegóły książki!

Usuwanie powtórzeń kodu (DRY!)

Aby wyświetlić imię i nazwisko autora, w plikach `books_controller.rb` oraz `show.html.erb` użyliśmy podobnego kodu. Było to po prostu połączenie imienia i nazwiska w jeden łańcuch.

Aby usunąć to powtórzenie zdefiniujemy nową metodę w modelu `Author`, która będzie działać jak wirtualny atrybut.

Dodajmy zatem metodę `full_name` w pliku `app/models/author.rb`:

```
class Author < ActiveRecord::Base
  def full_name
    "#{self.first_name} #{self.last_name}"
  end
end
```

Usuwanie powtórzeń

Następnie w `show.html.erb` zastępujemy:

```
<%= @book.author.first_name + ' ' + @book.author.last_name %>
```

przez:

```
<%= @book.author.full_name %>
```

A w `books_controller.rb` zastępujemy:

```
@authors = Author.find(:all).map do |author|  
  [ author.first_name + ' ' + author.last_name, author.id]  
end
```

przez:

```
@authors = Author.find(:all).map do |author|  
  [ author.full_name, author.id]  
end
```

Teraz sprawdzamy czy wszystko działa jak należy!

Uzupełnienie związku w modelu Author

Zaglądamy do `app/models/book.rb`:

```
class Book < ActiveRecord::Base
  belongs_to :author
end
```

Dzięki zastosowaniu `author:references` przy generowaniu szkieletu, zdefiniowana jest relacja wiele-do-jednego pomiędzy książką a autorem.

Informację o tej relacji musimy jednak również dodać do modelu `author`:

```
class Author < ActiveRecord::Base
  has_many :books
end
```

Zadanie

- ▶ Zmodyfikuj listing książek tak aby zawierał imię i nazwisko autora!
- ▶ Zmodyfikuj szczegóły autora tak aby zawierały listę napisanych przez niego książek!
- ▶ Zaimplementuj aplikację portfela walut z wykorzystaniem Ruby on Rails.

Ruby on Rails – materiały

Dokumentacja:

- ▶ <http://apohllo.pl/dydaktyka/interfejsy/rails>
- ▶ <http://guides.rubyonrails.org/> – Rails Guides
- ▶ <http://apohllo.pl/guides/index.html> – Rails Guides (po polsku, omawia Rails 2.3.5)
- ▶ <http://railsapi.com> – działa szybko i jest aktualna

Edytory:

- ▶ Vim + Rails plugin (w pracowniach)
- ▶ Aptana RadRails (na bazie Eclipse)
- ▶ NetBeans + Ruby plugin
- ▶ RubyMine (płatna)