

EPI: Interfejs Graficzny 2010/2011

Podstawy Ruby on Rails

– ciąg dalszy

Aleksander Pohl

26 stycznia 2011

Plan prezentacji

Wstęp

Warstwa modelu

Warstwa kontrolera

Warstwa widoku

Trasowanie

Powtórzenie

Ostatnim razem zbudowaliśmy prostą bibliotekę i zobaczyliśmy RoR w działaniu:

- ▶ komendę rails
- ▶ serwer WEBrick
- ▶ zarządzanie schematem bazy danych z użyciem migracji
- ▶ zarządzanie zawartością bazy danych z użyciem rusztowania
- ▶ architekturę MVC w Rails
 - ▶ modele
 - ▶ widoki
 - ▶ kontrolery
- ▶ trasowanie

Dzisiaj przyjrzymy się szczegółom!

komenda rails

```
$ rails library_ini
  create
  create  app/controllers
  create  app/helpers
  create  app/models
  create  app/views/layouts
  create  config/environments
  create  db
  create  doc
  create  lib
  create  lib/tasks
  create  log
  [...]
```

Serwer aplikacji

```
$ rails server
=> Booting Mongrel
=> Rails 3.0.0 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

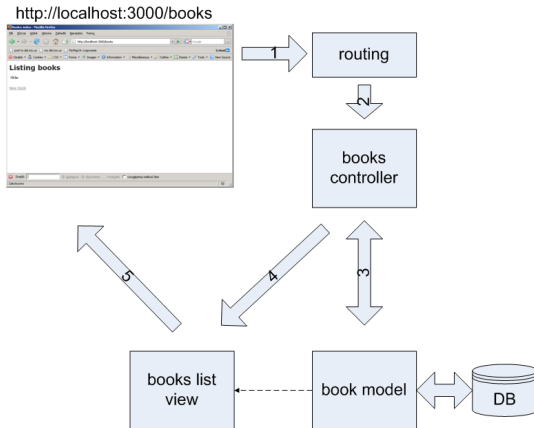
- ▶ domyślny port to 3000
- ▶ jeśli chcemy uruchomić więcej instancji na tym samym porcie pojawia się błąd – Address already in use - bind(2) (Errno::EADDRINUSE)
- ▶ aby wybrać inny port używamy **opcji -p**, np.:
`rails server -p 4000`

Serwer aplikacji – cd.

Railsy mogą również pracować z Apache, Lighttpd, Mongrel, Thinem i innymi serwerami ogólnego przeznaczenia.

Od niedawna dostępny jest moduł dla Apache **mod_passenger**, który sprawia, że uruchomienie Railsów jest prawie tak proste jak aplikacji napisanej w PHP.

MVC w RoR



Rusztowania

- ▶ Rusztowanie (ang. scaffold) – implementuje **funkcjonalności CRUD** (create, read, update, destroy) dla określonego modelu.
- ▶ W Rails od wersji 2.x generowanie rusztowania (tzw. rusztowanie **statyczne**) może być połączone z generowaniem **modelu** oraz **tras**.
- ▶ Generowanie rusztowania pozwala zatem utworzyć elementy należące do wszystkich warstw aplikacji.
- ▶ Akcje tworzone dla rusztowania: `list`, `show`, `new`, `create`, `edit`, `update`, `destroy`

Rusztowanie dynamiczne

- ▶ Standardowe rusztowanie **dynamiczne** zostało usunięte z Rails i dostępne jest jedynie jako plugin.
- ▶ **ActiveScaffold** (Rails 2.x) oraz **rails_admin** (Rails 3.x) są jednak znacznie lepszymi rozwiązaniami:
 - ▶ dynamiczne generowanie funkcjonalności CRUD (jw.)
 - ▶ wiele opcji konfiguracyjnych
 - ▶ wykorzystywany w panelach administracyjnych, które nie wymagają wyszukanego interfejsu użytkownika
 - ▶ <http://activescaffold.com/>
 - ▶ http://github.com/sferik/rails_admin/

Rusztowanie statyczne

- ▶ rails generate scaffold model_name field1:type1 field2:type2 ...

- ▶ przykład:

```
rails generate scaffold book title:string pages:integer
...
create app/views/books
create app/views/books/index.html.erb
...
create public/stylesheets/scaffold.css
create app/controllers/books_controller.rb
create test/functional/books_controller_test.rb
create app/helpers/books_helper.rb
route map.resources :books
dependency model
...
create app/models/book.rb
create test/unit/book_test.rb
create test/fixtures/books.yml
create db/migrate
create db/migrate/20081110010308_create_books.rb
```

- ▶ wszystkie pliki są generowane – muszą być modyfikowane po zmianie modelu

ActiveScaffold

- ▶ w kontrolerze

```
active_scaffold "nazwa_modelu"
```

- ▶ przykład:

```
class BooksController < ApplicationController
```

```
  active_scaffold "book" do |config|
    config.list.columns = [:title]
    config.list.per_page = 30
    config.list.label = "Ksiazki"
  end
```

```
end
```

- ▶ żadne pliki nie są generowane, wszystkie metody są dynamiczne
- ▶ rusztowanie dostosowuje się do zmian w modelu
- ▶ interfejs użytkownika oparty na technologii Ajax

SVN – kontrola wersji

- ▶ `svn checkout (co)`
`svn+ssh://login@wierzba.wzks.uj.edu.pl/ig/login` –
kopiowanie repozytorium
- ▶ `svn update (up)` – ściąganie na komputer lokalny ostatnich
zmian z repozytorium
- ▶ `svn commit (ci) -m "komunikat"` – przesłanie własnych
zmian do repozytorium z odpowiednim komunikatem
- ▶ `svn add plik` – dodanie nowego pliku do repozytorium
- ▶ `svn rm plik` – usunięcie pliku z repozytorium
- ▶ `svn state (st)` – wylistowanie zmian i plików poza kontrolą
wersji
- ▶ `svn log` – wyświetlenie historii zmian

SVN – typowa sesja

- ▶ `svn co svn+ssh://...`
- ▶ edycja plików, dodanie nowych plików
- ▶ `svn up` – żeby upewnić się czy mamy najnowszą wersję
- ▶ `svn st`
- ▶ `svn add x y z`
- ▶ `svn ci -m "komunikat"`
- ▶ mija dzień ...
- ▶ `svn up` – ściągamy najnowszą wersję
- ▶ edycja plików, itd.

Plan prezentacji

Wstęp

Warstwa modelu

Warstwa kontrolera

Warstwa widoku

Trasowanie

Model – ORM

Dynamiczna implementacja podstawowych operacji na tabeli w bazie danych – CRUD.

app/models/book.rb

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

▶ konstruktory – create:

- ▶ book = Book.new(:title => 'some title', :author_id => 1)
- ▶ book = Book.new

▶ atrybuty – read:

- ▶ book.title
- ▶ book.title = 'new title'

▶ book.save

– create, update

- ▶ book.update_attributes(:title => 'new title')

– update

- ▶ book.destroy

Model – 'dynamiczne' odszukiwanie obiektów

- ▶ za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

- ▶ jw. (wersja bezpieczna):

```
Book.find_by_id(1)
```

- ▶ przez wartość atrybutu (zwraca tablicę):

```
Book.find_all_by_title('Ruby')
```

- ▶ przez wartość atrybutu (zwraca obiekt lub nil):

```
Book.find_by_title('Ruby')
```

- ▶ z opcjami (zwraca tablicę):

```
Author.find(:all, :conditions =>  
  ['last_name=?', last_name_var], :order => 'first_name ASC')
```

- ▶ z opcjami (zwraca obiekt lub nil):

```
Author.find(:first, :conditions =>  
  ['last_name=?', last_name_var], :order => 'first_name ASC')
```

rails console – konsola Rails

```
$ rails console
```

```
Loading development environment.
```

```
irb(main):001:0> authors = Author.find(:all)
=> [#<Author:0x48920c0 @attributes={"id"=>"1", "first_name"=>"Charles", "last_name"=>"Dickens"}>]
irb(main):002:0> a = Author.new(:first_name => 'George', :last_name => 'Eliot')
=> #<Author:0x488dfac @new_record=true, @attributes={"first_name"=>"George", "last_name"=>"Eliot"}>
irb(main):003:0> Author.find(:all)
=> [#<Author:0x48891a0 @attributes={"id"=>"1", "first_name"=>"Charles", "last_name"=>"Dickens"}>]
irb(main):004:0> a.save
=> true
irb(main):005:0> Author.find(:all)
=> [#<Author:0x48838a4 @attributes={"id"=>"1", "first_name"=>"Charles", "last_name"=>"Dickens"}>,
#<Author:0x4883840 @attributes={"id"=>"2", "first_name"=>"George", "last_name"=>"Eliot"}>]
irb(main):006:0> Author.find(1)
=> #<Author:0x4890cac @attributes={"id"=>"1", "first_name"=>"Charles", "last_name"=>"Dickens"}>
irb(main):007:0> Author.find(1000)
ActiveRecord::RecordNotFound: Couldn't find Author with ID=1000
  from /usr/lib/ruby/gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:1031:in 'find_one'
  from /usr/lib/ruby/gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:1014:in 'find_from_ids'
  from /usr/lib/ruby/gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:419:in 'find'
  from (irb):2
irb(main):008:0> Author.find_by_id(1000)
=> nil
irb(main):009:0> quit
```

Model – związki

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

Obiekty klasy 'Book' posiadają teraz następujące dodatkowe metody:

```
book.author
book.author=(author)
book.author?(some_author)
book.author.nil?
book.build_author(...)
book.create_author(...)
```

Model – związki – cd.

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

Obiekty klasy 'Author' posiadają teraz następujące dodatkowe metody:

```
author.books
author.books<<(book)
author.books.delete(book)
author.books=[book1,book2,...]
author.book_ids
author.book_ids=[id1,id2,...]
author.books.clear
author.books.empty?
author.books.size
author.books.find(...)
author.books.build(...)
author.books.create(...)
```

Zadanie!

W konsoli Rails

- ▶ Tworzymy i zapisujemy autora „Henryk Sienkiewicz”
- ▶ Tworzymy i zapisujemy książkę „W pustyni i w puszczy”, której autorem jest H. Sienkiewicz
- ▶ Tworzymy i zapisujemy książkę „Quo vadis?”, autor j.w.
- ▶ Tworzymy i zapisujemy autora „Adam Mickiewicz”
- ▶ Tworzymy i zapisujemy książkę „Pan Tadeusz”, której autorem jest A. Mickiewicz
- ▶ Tworzymy i zapisujemy książkę „Sonety krymskie”, autor j.w.
- ▶ Wyszukujemy wszystkich autorów o imieniu „Adam”
- ▶ Wyszukujemy pierwszego autora o imieniu „Adam” i wyświetlamy tytuł jego pierwszej książki (puts).
- ▶ Wyszukujemy książkę o tytule „Sonety krymskie” i wypisujemy nazwisko jej autora.

Zadanie!

Modyfikujemy bibliotekę w taki sposób, aby strona zawierająca szczegóły autora wyświetlała listę jego książek.

- ▶ `http://localhost:3000/authors/show/some_id`
- ▶ Podpowiedź: autor może napisać wiele książek ;-)

Walidacje modelu

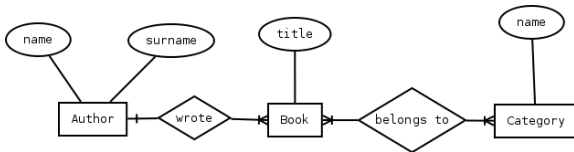
```
class Person < ActiveRecord::Base
  validates_uniqueness_of :user_name, :scope => :account_id
  validates_presence_of :first_name
  validates_numericality_of :shoe_size, :on => :create
  validates_length_of :first_name, :maximum => 30
  validates_length_of :user_name, :within => 6..20,
    :too_long => "pick a shorter name",
    :too_short => "pick a longer name"
  validates_format_of :email,
    :with => /\A([^\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i,
    :on => :create
  validates_confirmation_of :email_address,
    :message => "should match confirmation"
  validates_acceptance_of :terms_of_service
end
```

Zadanie!

Dodajemy walidacje do modelu 'Author' oraz 'Book'!

- ▶ książka musi posiadać tytuł i autora
- ▶ autor musi posiadać imię i nazwisko
- ▶ może istnieć tylko jeden autor z danym imieniem i nazwiskiem

Model i baza danych



```

class Author < ActiveRecord::Base
  has_many :books
end
class Book < ActiveRecord::Base
  belongs_to :author
  has_and_belongs_to_many :categories
end
class Category < ActiveRecord::Base
  has_and_belongs_to_many :books
end
  
```

Model i baza danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`
2. w bazie danych wszystkie tabele zaczynają się z małej litery i są w liczbie mnogiej, np. `authors`, `books`, `categories`
3. związek jeden-do-wiele:
 - 3.1 po stronie „jeden” nazwa związku w liczbie mnogiej, np. `author.books`, brak pola w bazie danych
 - 3.2 po stronie „wiele” nazwa związku w liczbie pojedynczej, np. `book.author`, nazwa pola w bazie: nazwa powiązanego modelu + `_id`, np. `author_id`

Model i baza danych

4. związek wiele-do-wiele:

- 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
- 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
- 4.3 tabela złączeniowa nie musi posiadać klucza id, powinna posiadać klucz unikalny obejmujący oba klucze obce

Migracje 1/3

- ▶ pozwalają na inkrementacyjne budowanie schematu bazy danych
- ▶ pliki migracji – `db/migrate/*`
 - ▶ pusty plik migracji generowany jest przez polecenie:
rails generate migration migration_name
 - ▶ w poleceniu można dodać pary **nazwa_pola:typ**, które zostaną automatycznie uwzględnione w migracji
 - ▶ plik migracji opisuje zmianę schematu bazy danych
 - ▶ 2 metody: **up** i **down** opisują zmianę schematu i jej odwrotność; pozwala to na wprowadzanie i wycofywanie zmian schematu
 - ▶ posiadają numer, który w Rails 2.0 jest generowany na podstawie czasu powstania pliku; dzięki temu możliwe jest bezproblemowe generowanie migracji przez wielu programistów pracujących niezależnie

Migracje 2/3

- ▶ `schema_info`
 - ▶ `schema_info` jest tabelą w bazie danych posiadającą jedno pole tekstowe – **version**
 - ▶ zawiera numery wszystkich wprowadzonych migracji
- ▶ `$ rake db:migrate`
 - ▶ **rake db:migrate** sprawdza numery w bazie danych i porównuje je z numerami migracji w katalogu **db/migrate**
 - ▶ jeśli w bazie danych brakuje jakiś numerów migracji, są one aplikowane w kolejności odpowiadającej rosnącemu numerom
 - ▶ w celu wycofania zmian schematu bazy danych aż do określonej wersji wprowadzamy
`rake db:migrate VERSION=timestamp`
powodzenie tego procesu zależy od poprawnej implementacji metody `down`
- ▶ <http://guides.rubyonrails.org/migrations.html>

Migracje 3/3

Inne przydatne polecenia:

- ▶ **db:create** – tworzy bazę danych dla środowiska deweloperskiego
- ▶ **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację
- ▶ **db:migrate:reset** – czyści bazę i od początku przeprowadza wszystkie migracje
- ▶ **db:rollback** – wycofuje ostatnią migrację
- ▶ **db:version** – wyświetla numer ostatniej wprowadzonej migracji

Model i migracje 1/3

```
class CreateAuthors <
  ActiveRecord::Migration
  def self.up
    create_table :authors do |t|
      t.string :name
      t.string :surname
    end
  end

  def self.down
    drop_table :authors
  end
end
```

```
class CreateBooks <
  ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.string :title
      t.references :author
    end
  end

  def self.down
    drop_table :books
  end
end
```

Model i migracje 2/3

```
class CreateCategories < ActiveRecord::Migration
  def self.up
    create_table :categories do |t|
      t.string :name
    end
  end

  def self.down
    drop_table :categories
  end
end
```

Model i migracje 3/3

```
class CreateBooksCategories < ActiveRecord::Migration
  def self.up
    create_table :books_categories, :id => false do |t|
      t.references :book
      t.references :category
    end
    add_index :books_categories, [:book_id, :category_id]
  end

  def self.down
    drop_table :books_categories
  end
end
```

Model – więcej informacji

- ▶ <http://guides.rubyonrails.org/migrations.html> – **migracje** schematu bazy danych
- ▶ http://guides.rubyonrails.org/active_record_querying.html – **wyszukiwanie** obiektów w bazie danych
- ▶ http://guides.rubyonrails.org/association_basics.html – **asocjacje** (związki) modeli
- ▶ http://guides.rubyonrails.org/active_record_validations_callbacks.html – **walidacje**

Plan prezentacji

Wstęp

Warstwa modelu

Warstwa kontrolera

Warstwa widoku

Trasowanie

Kontroler

Reaguje na żądanie poprzez wykonanie akcji, renderuje widok lub przekierowuje do innej akcji.

```
/app/controllers/welcome_controller.rb
```

```
class WelcomeController < ApplicationController
  def index
    @message = 'Hello World!'
  end

  def hello_render
    @message = 'Hello Rails!'
    render :action => 'index'
  end

  def hello_redirect
    redirect_to :action => 'index'
  end
end
```

```
/app/views/welcome/index.html.erb
```

```
<n><h><%= @message %></h></n>
```



Kontroler – przepływ 1 – domyślna akcja

1. `http://host:port/welcome`
2. wykonuje domyślną akcję `index` w kontrolerze `WelcomeController`
3. renderuje domyślny widok dla akcji `index` – `views/welcome/index.html.erb`

Kontroler – przepływ 2 – renderowanie innej akcji

1. `http://host:port/welcome/hello_render`
2. wykonuje akcję `hello_render` w kontrolerze `WelcomeController`
3. renderuje widok określony dla akcji `index`, ale jako komunikat wyświetla się `'Hello Rails!'`

Kontroler – przepływ 3 – przekierowanie

1. `http://host:port/welcome/hello_redirect`
2. wykonuje akcję `hello_redirect` kontrolera `WelcomeController`
3. akcja `hello_redirect` przekierowuje do akcji `index`

Renderowanie vs. przekierowanie

renderowanie akcji:

- ▶ przesłanie treści do użytkownika
- ▶ `render :action` – użycie szablonu powiązanego z daną akcją
- ▶ domyślnie – użycie szablonu nazwanego tak jak akcja
- ▶ wszystkie zmienne instancyjne ustawione w pierwszej akcji są dostępne w widoku
- ▶ żadne dodatkowe zmienne instancyjne nie są ustawiane

przekierowanie do akcji:

- ▶ podmiana akcji
- ▶ wszystkie zmienne instancyjne oraz parametry żądania ustawione w pierwszej akcji są tracone po przekierowaniu
- ▶ tylko zmienne instancyjne ustawione w drugiej akcji są dostępne
- ▶ obiekt `flash` jest jednak widoczny również po przekierowaniu

DoubleRenderError

Wyjątek `DoubleRenderError` zostanie rzucony jeśli aplikacja opuszcza akcję, w której wielokrotnie zażądano renderowania. Przykład:

```
def do_something
  redirect_to :action => "elsewhere" #1st render in 'elsewhere'
  render :action => "overthere" #2nd render, DoubleRenderError
end
```

Jeśli trzeba renderować odmienne widoki zależnie od jakiegoś warunku, należy użyć wyrażenia `and return` po `render/redirect`:

```
def do_something
  if status.nil?
    redirect_to(:action => "elsewhere") and return
  end
  render :action => "overthere"
end
```

Kontroler + Widok

Akcja kontrolera powoduje renderowanie widoku – domyślnie jego nazwa jest taka sama jak nazwa akcji, która odpowiedziała na żądanie (lub ostatnia, do której żądanie zostało przekierowane)
Kontroler i widok dzielą zmienne instancyjne kontrolera (zaczynające się od znaku '@')

app/controllers/hello_controller.rb:

```
def hello
  @msg='Hello World!'
end
```

app/views/hello/hello.html.erb:

```
<p><b><%= @msg %></b></p>
```

Dzielą również specjalne zmienne:

- ▶ session – przechowuje dane sesji użytkownika
- ▶ params – przechowuje parametry aktualnego żądania
- ▶ flash – przechowuje tymczasowe obiekty akcji

Kontroler + widok – negocjacja treści

Kontroler może użyć różnej reprezentacji danych (HTML, XML, JSON, itp.), w zależności od wystosowanego żądania

app/controllers/books_controller.rb

```
class BooksController < ApplicationController
  # GET /books/1
  # GET /books/1.xml
  def show
    @book = Book.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.xml { render :xml => @book }
    end
  end
end
```

Dla każdego formatu używany jest domyślnie widok o nazwie:
nazwa_akcji.nazwa_formatu.nazwa_renderera, np.
show.html.erb, show.xml.haml

Negocjacja treści – cd.

- ▶ /books/1

```
<p>
  <b>Title:</b>
  Alef
</p>
<p>
  <b>Author:</b>
  Jorge Luis Borges
</p>
```

- ▶ /books/1.xml lub /books/1 Accept:application/xml

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

Plan prezentacji

Wstęp

Warstwa modelu

Warstwa kontrolera

Warstwa widoku

Trasowanie

Widok – erb

Erb (zagnieżdżony Ruby)

```
<%= some_statement %>
```

– ewaluuje wyrażenie, zamienia je na łańcuch znaków i wyświetla w miejscu wystąpienia

```
<% some_statement %>
```

– ewaluuje wyrażenie, ale nie wyświetla wyniku

Przykład:

```
<b>Names of all the people</b>  
<% for person in @people %>  
  Name: <%= person.name %><br/>  
<% end %>
```

Widok – haml

haml jest alternatywą dla erb. Pozwala na tworzenie znacznie bardziej zwięzłego kodu:

```
#people
  %b Names of all the people
  - for person in @people
    .name
      Name:
      = person.name
```

jest zamieniane na:

```
<div id="people">
  <b>Names of all the people</b>
  <div class="name">
    Name: John Brown
  <div>
  <div class="name">
    Name: Ann Black
  <div>
</div>
```

Widok – układ strony (layout)

app/views/layouts/library.html.erb:

```
<html>
<head>
  <title>Library</title>
  <%= stylesheet_link_tag "scaffold" %>
</head>
<body>
  <%= link_to "Books", :controller => "books" %>
  <%= link_to "Authors", :controller => "authors" %>
  <p style="color: green"><%= flash[:notice] %></p>

  <%= yield %>

</body>
</html>

class AuthorsController < ApplicationController
  layout "library"
  ...
end
```



Zadanie!

- ▶ Dodaj layout do aplikacji library!

Helpery

- ▶ pozwalają na zminimalizowanie ilości kodu Rubiego w widoku
- ▶ są modułami (a nie klasami) Rubiego
- ▶ Railsy posiadają wiele helperów wbudowanych (np. do tworzenia linków, załączania obrazów, etc.)
- ▶ użytkownik może definiować własne helpery w katalogu `app/helpers`

Wbudowane helpery Rails

Wbudowane helpery Rails, np. `link_to`:

```
<%= link_to 'Back', books_path %>
```

Wynik:

```
<a href="/books/list">Back</a>
```

Inne:

- ▶ `h` – escapowanie tagów HTML (obecnie domyślnie włączony)
- ▶ `raw` – brak escapowania tagów HTML
- ▶ `form_for` – szkielet formularza
- ▶ `text_field`, `check_box`, `label`, `password_field`, `radio_button`, `text_area`, `select` – pola formularza
- ▶ `auto_link` – automatycznie tworzy linki do adresów wykrytych w tekście
- ▶ `simple_format` – proste formatowanie tekstu
- ▶ `ActionView::Helpers` – pozostałe helpery

Formularze

- ▶ helper `form_for` pozwala w łatwy sposób tworzyć formularz dla określonego obiektu
- ▶ ma on postać bloku z jednym argumentem
- ▶ argument bloku pozwala na tworzenie pól formularza powiązanych z odpowiednimi atrybutami obiektu

```
<%= form_for @book do |f| %>
<p>
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</p>
<p>
  <%= f.submit "Create" %>
</p>
<% end %>
```

Formularze dla powiązanych obiektów

- ▶ możliwe jest łatwe tworzenie formularzy dla obiektów powiązanych relacjami jeden-do-jeden lub wiele-do-wiele
- ▶ np. jeśli chcemy jednocześnie utworzyć książkę i jej autora
- ▶ książka musi akceptować zagnieżdżone parametry dla autora:

```
class Book < ActiveRecord::Base
  belongs_to :author
  accepts_nested_attributes_for :author,
    :reject_if => :all_blank
end
```

- ▶ w formularzu korzystamy z helpera **fields_for**
- ▶ jego argumentem może być powiązany obiekt (np. `@book.author`) lub symbol (np. `:author`)
 - ▶ w przypadku obiektu – musi on istnieć
 - ▶ w przypadku symbolu – zostanie utworzony nowy obiekt, jeśli powiązany obiekt nie istnieje

Formularze dla powiązanych obiektów

zmodyfikowany formularz pozwala na wybranie istniejącego lub utworzenie nowego autora

```
<%= form_for(@book) do |f| %>
  <!-- pola książki ... -->
  <div>
    <%= f.fields_for :author do |author_f| %>
      <p>
        <%= author_f.label :first_name %>
        <%= author_f.text_field :first_name %>
      </p>
      <p>
        <%= author_f.label :last_name %>
        <%= author_f.text_field :last_name %>
      </p>
    <% end %>
  </div>
  <!-- submit ... -->
<% end %>
```

Helperzy użytkownika

- ▶ dla widoków określonego kontrolera, ładowane są następujące helperzy użytkownika:
 - ▶ ApplicationHelper **app/helpers/application_helper.rb** zawierający metody pomocnicze wykorzystywane we wszystkich kontrolerach
 - ▶ NazwaKontroleraHelper np. dla
app/controllers/**authors_controller.rb**
app/helpers/**authors_helper.rb** zawierający metody pomocnicze wykorzystywane tylko w widokach tego kontrolera
- ▶ dodatkowe helperzy można aktywować w kontrolerze:

```
class AuthorsController < ApplicationController
  helper BooksHelper
end
```

Helperzy użytkownika – prosty przykład

Definicja w `helpers/books_helper.rb`:

```
module BooksHelper
  def book_fancy_title(book)
    raw("<b>***#{h(book.title)}***</b>")
  end
end
```

Użycie w `views/books/show.html.erb`:

```
<%= book_fancy_title(@book) %>
```

Wynik:

```
<b>***Mały Książę***</b>
```

Helepry użytkownika – rozbudowany przykład

Definicja w `app/helpers/application_helper.rb`:

```
module ApplicationHelper
  def close()
    raw link_to(image_tag("cancel.png", :title => "zamknij",
                        :class => "close_button"), "#")
  end

  def panel(&block)
    content_tag(:div,
      content_tag(:div, :class => "content",
                  :style => "width:95%", &block) +
      content_tag(:div, close(), :class => "actions") +
      content_tag(:div, tag(:span), :class => "clear") +
      '</div>', :html_safe, :class => "relations")
  end
end
```

Helepry użytkownika – rozbudowany przykład

Użycie:

```
<%= panel do %>
  <p> Jakiś tekst </p>
<% end %>
```

Wynik:

```
<div class="relations"><div class="content" style="width:95%">
  <p> Jakiś tekst </p>
</div>
<div class="actions">
  <a href="#">
    
  </a>
</div>
<div class="clear"><span /></div></div></div>
```

Fragmenty widoków (partials)

- ▶ ich nazwa rozpoczyna się od znaku podkreślenia
- ▶ pozwalają na reużytkowanie fragmentów widoku w wielu widokach.

new.html.erb

```
<h1>New book</h1>
<% form_for(@book) do |f| %>
  <%= render :partial => 'form',
    :object => f %>
<p>
  <%= f.submit "Create" %>
</p>
<% end %>
<%= link_to 'Back', books_path %>
```

edit.html.erb

```
<h1>Editing book</h1>
<% form_for(@book) do |f| %>
  <%= render :partial => 'form',
    :object => f %>
<p>
  <%= f.submit "Update" %>
</p>
<% end %>
<%= link_to 'Show', @book %> |
<%= link_to 'Back', books_path %>
```

Fragmenty widoków cd.

▶ _form.html.erb

```
<p>
  <%= form.label :title %><br />
  <%= form.text_field :title %>
</p>
<p>
  <%= form.label :author %><br />
  <%= form.select :author_id, @authors %>
</p>
```

- ▶ domyślnie fragment jest szukany w katalogu widoków danego kontrolera, czyli np. dla BooksController w app/views/books.
- ▶ jeśli chcemy skorzystać z fragmentu innego kontrolera, musimy poprzedzić jego nazwę nazwą kontrolera, np.

```
<%= render :partial => "books/form" %>
```

Zadanie!

Zastąp nazwisko autora wyświetlane w widoku książki przez link do widoku autora (`authors/show`)!

To samo zrób dla tytułów książek w widoku autora – niech będą linkami prowadzącymi do widoku szczegółów książki.

Rozwiązanie

W `views/books/show.html.erb` zastąp:

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

przez:

```
<%= link_to @book.author.full_name, author_path(@book.author) %>
```

W `views/authors/show.html.erb` zastąp:

```
<%= book.title %>
```

przez:

```
<%= link_to book.title, book_path(book) %>
```

Plan prezentacji

Wstęp

Warstwa modelu

Warstwa kontrolera

Warstwa widoku

Trasowanie

Trasowanie

- ▶ pozwala uniknąć ręcznego konstruowania linków wewnątrz aplikacji
- ▶ pozwala na stosowanie przyjaznych adresów URL
- ▶ zapewnia spójność adresów (dla określonej akcji zawsze tworzony jest ten sam link)
- ▶ pozwala budować aplikację w oparciu o koncepcję zasobów REST
- ▶ definiowane jest w pliku **config/routes.rb**

Trasowanie – definicja

```

Library::Application.routes.draw do
  resources :books          # wiele zasobów
  resources :authors do    # zasoby zagnieżdżone
    resources :books
  end
  resource :user_session   # jeden zasób
  match 'books/index'      # określony kontroler i akcja
  match 'login' => "user_sessions#new"
                          # zmiana adresu URL
  match 'logout' => "user_sessions#destroy", :as => :logout
                          # zmiana adresu oraz
                          # utworzenie helpera 'logout_path'
  get 'books/new'          # ograniczenie do HTTP 'get'
  root :to => 'books#index' # strona startowa
  match ':controller(/:action(/:id))'
                          # reguła 'catch all'
end

```

Trasowanie – użycie

```
<%= link_to "książki", :controller => "books", :action => "index" %>
  <a href="/books">książki</a>
<%= link_to "książki", books_path %>
  <a href="/books">książki</a>
<%= link_to @book.title, @book %>
  <a href="/books/1">Mały Książę</a>
<%= link_to @author.books.first.title, [@author, @author.books.first] %>
  <a href="/authors/2/books/1"/>Mały Książę</a>
<%= link_to "wyloguj", logout_path %>
  <a href="/logout"/>wyloguj</a>
<%= link_to "start", root_url %>
  <a href="/">start</a>
```

Trasowanie – wyświetlenie dostępnych tras

```
rake routes
books GET /books(:format)
:action=>"index", :controller=>"books"
books POST /books(:format)
:action=>"create", :controller=>"books"
new_book GET /books/new(:format)
:action=>"new", :controller=>"books"
edit_book GET /books/:id/edit(:format)
:action=>"edit", :controller=>"books"
```

Konfiguracja strony startowej

Dotychczas domyślną stroną naszej aplikacji był ekran powitalny Rails. Zamieńmy go na listing książek używając do tego mechanizmu trasowania.

1. **kasujemy plik `public/index.html`** – jest to ekran powitalny Rails
2. otwieramy plik **`config/routes.rb`**
3. dodajemy linię:

```
root :to => "books#index"
```

Upewniamy się, że **usunęliśmy plik `public/index.html`**!

Teraz `localhost:3000` przekierowuje użytkownika do listingu książek.

Podziękowania

Dla:

- ▶ Agnieszki Figiel, za udostępnienie prezentacji w postaci plików źródłowych
- ▶ Marka Kowalcze oraz Jakuba Kuźmy z grupy SRUG (srug.pl), za pomoc przy kolorowaniu składni w Latex'u