Laboratorium nr 3
Podstawy Ruby on Rails

Aleksander Smywiński-Pohl

Elektroniczne Przetwarzanie Informacji

Plan prezentacji

Utworzenie aplikacji

Author

Book

Końcowe porządki

System do obsługi biblioteki

Bardzo prosta biblioteka. Funkcjonalności:

1. zarządzanie autorami – CRUD
2. zarządzanie książkami – CRUD
3. uwzględnienie związku między autorami i książkami

Plan działania

1. **wygenerowanie struktury aplikacji**
   - ▷ `rails new library`
2. **uruchomienie serwera**
   - ▷ `rails server`
3. **konfiguracja bazy danych (opcjonalne)**
4. **dodanie rusztowania** `Author`
   - ▷ `rails generate scaffold author ..`
5. **dodanie rusztowania** `Book`
   - ▷ `rails generate scaffold book ..`
6. **dodanie wyboru autora przy edycji książki**
7. **powiązanie modeli** `Book` **i** `Author`

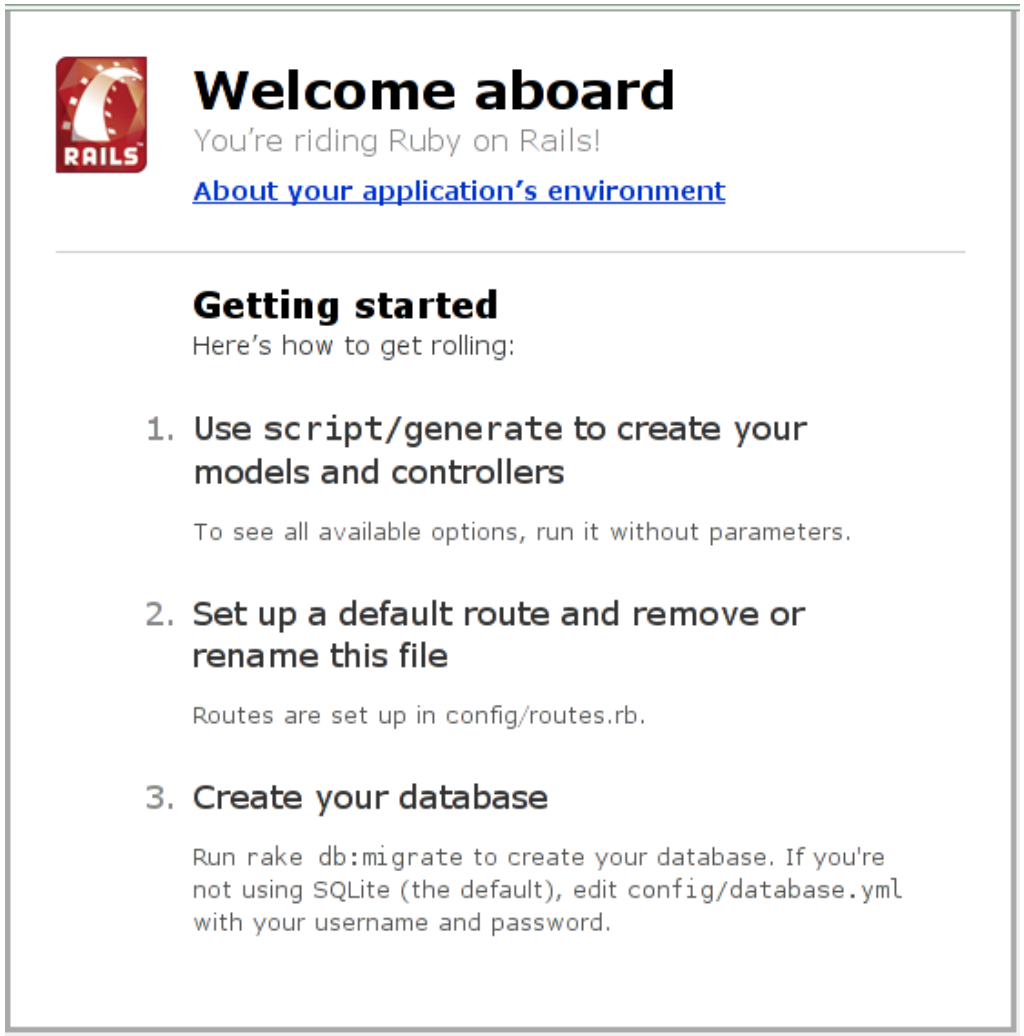
   ```
   has_many :books
   ```
8. **uwzględnienie autora przy wyświetlaniu książki**
9. **usunięcie powtórzeń kodu**

Tworzenie nowego projektu

1. przejdź do katalogu, w którym ma znaleźć się główny katalog projektu
2. ▷ `rails new library`[1]
3. ▷ `cd library`
4. w pliku `Gemfile` odkomentowujemy wpis
   `gem 'therubyracer'`
5. ▷ `bundle`
6. ▷ `rails server -p 3300`
   - domyślny numer portu to 3000; możemy zmienić go za pomocą opcji -p
7. otwórz przeglądarkę i wprowadź adres
   `http://localhost:3300`

[1] ▷ oznacz znak zachęty – nie wprowadzamy go!

Rails – okno powitalne



Rails – struktura katalogów

- `app`: kod źródłowy
- `bin`: skrypty aplikacji (bundle, rake, rails)
- `config`: konfiguracja
- `db`: schemat bazy danych
- `lib`: dodatkowe biblioteki
- `log`: logi
- `public`: treści statyczne
- `test`: automatyczne testy
- `tmp`: pliki tymczasowe
- `vendor`: dodatki (pluginy)

Katalog 'app'

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

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

Konfiguracja bazy danych (opcjonalne)

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

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

Plan prezentacji

Utworzenie aplikacji

Author

Book

Końcowe porządki

Tworzenie rusztowania dla modelu `Author`

Przechodzimy do katalogu `libarary` i generujemy rusztowanie:

▷ `rails g scaffold author name:string surname:string`

- 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

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

Zaglądamy pod adres **http://localhost:3300/authors**

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 danych[2]
- **app/controllers/authors_controller.rb** – definiuje akcje `index`, `show`, `new`, `create`, `edit`, `update` oraz `destroy`
- **app/views/authors/*** – widoki w plikach html.erb

[2] **Uwaga:** nazwa modelu jest w liczbie pojedynczej a tabeli – w liczbie mnogiej.

Podgląd pliku migracji i modelu

db/migrate/xxx_create_authors.rb

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

      t.timestamps
    end
  end
end
```

app/models/author.rb

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

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, `name` varchar(255),
  `surname` varchar(255), `created_at` datetime,
  `updated_at` datetime
);
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
  before_action :set_author, only: [:show, :edit, :update, :destroy]

  # GET /authors
  # GET /authors.json
  def index
    @authors = Author.all
  end

  # GET /authors/1
  # GET /authors/1.json
  def show
  end

  # GET /authors/new
  def new
    @author = Author.new
  end

  # GET /authors/1/edit
  def edit
  end
  #...
end
```

Podgląd widoków i efekt końcowy

Otwieramy plik **app/views/authors/show.html.erb**.

```
<p id="notice"><%= notice %></p>
<p>
  <strong>Name:</strong>
  <%= @author.name %>
</p>
<p>
  <strong>Surname:</strong>
  <%= @author.surname %>
</p>
<%= link_to 'Edit', edit_author_path(@author) %> |
<%= link_to 'Back', authors_path %>
```

Plan prezentacji

Utworzenie aplikacji

Author

Book

Końcowe porządki

Tworzenie rusztowania dla modelu `Book`

W katalogu głównym (library):

▷ `rails g scaffold book title:string author:references`

następnie

▷ `rake db:migrate`

Oglądamy wynik pod adresem: **http://localhost:3300/books**

Coś jest jednak nie tak z autorem!

Zmiana sposobu tworzenia książek

Modyfikujemy plik **app/views/books/_form.html.erb**.
Zastępujemy:

```
<%= f.text_field :author_id %>
```

przez:

```
<%= f.select :author_id, @authors %>
```

Zmiana sposobu tworzenia książek – cd.

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

```
class BooksController < ApplicationController
  before_action :set_book, only: [:show, :edit, :update, :destroy]
  # tę linię dodajemy
  before_action :set_authors, only: [:new, :edit, :update, :create]

  # GET /books
  # ...
```

oraz na jego końcu (przed słowem kluczowym end!):

```
  def set_authors
    @authors = Author.all.map do |author|
      [ author.name + " " + author.surname, author.id]
    end
  end
```

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

Wyświetlanie szczegółów książki

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

```
<p id="notice"><%= notice %></p>

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

<p>
  <strong>Author:</strong>
  <%= @book.author.name + " " + @book.author.surname %>
</p>

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

Teraz oglądamy szczegóły książki!

Plan prezentacji

Utworzenie aplikacji

Author

Book

Końcowe porządki

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.name} #{self.surname}"
  end
end
```

Usuwanie powtórzeń – cd.

W modelu **book.rb** dodajemy również metodę **author_full_name**. Zabezpieczy nas ona przed książkami, które nie mają autora.

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

  def author_full_name
    if self.author
      self.author.full_name
    else
      "Autor nieznany"
    end
  end
end
```

Usuwanie powtórzeń – cd.

Następnie w **show.html.erb** zastępujemy:

```
<%= @book.author.name + " " + @book.author.surname %>
```

przez:

```
<%= @book.author_full_name %>
```

A w **books_controller.rb** zastępujemy:

```
@authors = Author.all.map do |author|
  [ author.name + " " + author.surname, author.id]
end
```

przez:

```
@authors = Author.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
  def full_name
    "#{self.name} #{self.surname}"
  end
end
```

Zadanie

- Zmodyfikuj listing książek tak aby zawierał imię i nazwisko autora.
- Zmodyfikuj widok szczegółów autora tak aby zawierał listę napisanych przez niego książek.
- Zmodyfikuj listing autorów, tak aby zawierał liczbę napisanych książek.

Ruby on Rails – materiały

Dokumentacja:

- `http://apohllo.pl/dydaktyka/interfejsy/rails`
- `http://guides.rubyonrails.org/` – Rails Guides
- `http://www.rubydoc.info/docs/rails/frames` – działa szykbo i jest aktualna

Edytory:

- Vim + Rails plugin (w pracowniach)
- Aptana RadRails
- NetBeans + Ruby plugin
- RubyMine (płatny)
- Sublime2 (płatny)

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 składniu w Latex'u