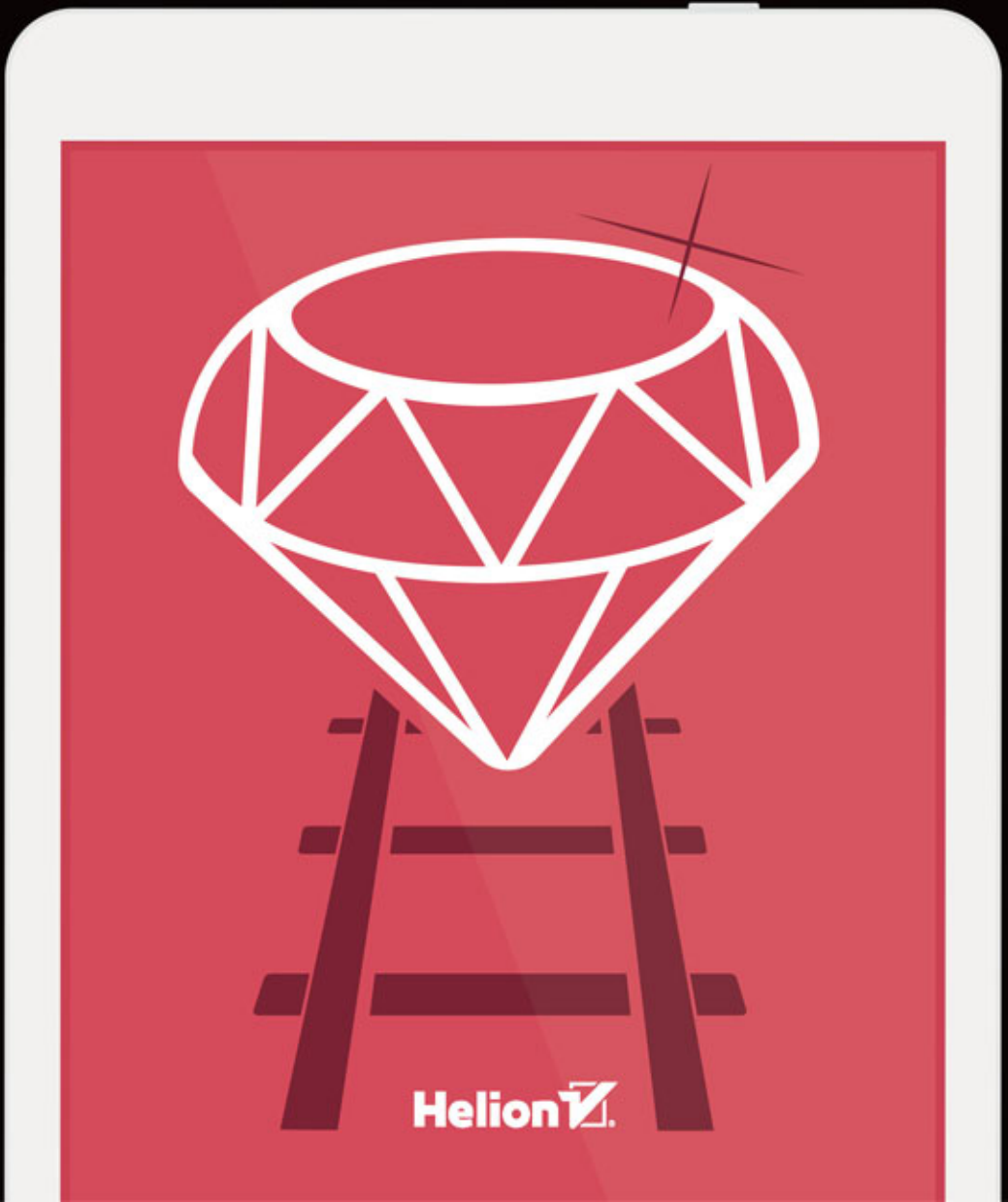
John Elder

RUBY ON RAILS

TWORZENIE APLIKACJI WWW



Tytuł oryginału: Learn Ruby On Rails For Web Development: Learn Rails The Fast And Easy Way!

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-1843-4

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/rrtwww
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

SPIS TREŚCI

Rozdział 4.

OBSŁUGA UŻYTKOWNIKÓW ZA POMOCĄ GEMA DEVISE

Twoja aplikacja się rozwija! Teraz, gdy poznałeś komponenty Bootstrap, możesz zacząć upiększać witrynę. Ale tak naprawdę… to aplikacja wciąż nic nie robi.

Czas to zmienić.

Jedną z kluczowych funkcjonalności aplikacji będzie możliwość tworzenia kont przez użytkowników, logowania, wylogowywania się i zmieniania profili.

Nie chcesz przecież, aby „piny" na Twojej stronie umieszczał przypadkowy użytkownik. W tym celu musi najpierw założyć konto i się zalogować.

Kiedyś napisanie kodu realizującego tę funkcjonalność przyprawiało o ból głowy i zabierało mnóstwo czasu. W dużej mierze wymagało to zaimplementowania obsługi baz danych, czego nienawidzę!

Na szczęście teraz jest Ruby on Rails, a do tego gem, który wykona za Ciebie całą mrówczą robotę. Gem nazywa się ***Devise***.

Jest to gem do uwierzytelniania użytkownika. Za jego pomocą można łatwo rejestrować użytkowników, umożliwiać im logowanie i wylogowywanie ze strony, jak również modyfikowanie swoich profili. Za pomocą tego gema tworzy się również formularze niezbędne do wykonania powyższych operacji, a dodatkowo realizuje całą skomplikowaną obsługę bazy danych.

Wystarczy jedynie zainstalować i skonfigurować gem i wszystko będzie od razu działać.

Otwórz stronę *RubyGems.org* i wyszukaj gem *Devise*. W chwili powstawania tej książki został on już pobrany i zainstalowany ponad 12 milionów razy. Jak widać, jest to popularny komponent (jakżeby mogło być inaczej — spróbuj wyobrazić sobie stronę WWW, która nie wymagałaby logowania i wylogowywania użytkowników!).

W chwili pisania tej książki dostępna była wersja 3.5.2 gema, którą należy dodać do pliku *Gemfile* w następujący sposób:

/Gemfile

```
 1 source 'https://rubygems.org'
 2 gem 'rails', '4.2.1'
 3 gem 'sass-rails', '~> 5.0'
 4 gem 'uglifier', '>= 1.3.0'
 5 gem 'coffee-rails', '~> 4.1.0'
 6 gem 'jquery-rails'
 7 gem 'turbolinks'
 8 gem 'jbuilder', '~> 2.0'
 9 gem 'sdoc', '~> 0.4.0', group: :doc
10 gem 'bootstrap-sass', '~> 3.3.5.1'
11 gem 'devise', '~> 3.5.2'
12
13 group :development, :test do
14   gem 'byebug'
15   gem 'web-console', '~> 2.0'
16   gem 'spring'
17 end
18
19 group :development, :test do
20   gem 'sqlite3'
21 end
22
23 group :production do
24   gem 'pg', '0.18.2'
25   gem 'rails_12factor', '0.0.3'
26 end
```

Jak zawsze, po dodaniu nowego gema do pliku *Gemfile* trzeba użyć polecenia `bundle install`:

```
1 $ bundle install
2
```

Oczywiście gem *Devise* wykonuje mnóstwo roboty, więc nie bądź zaskoczony, że jego instalacja i konfiguracja składa się z większej liczby kroków niż w przypadku innych gemów. Opisane są one w dokumentacji na stronie *RubyGems.org*, ale przeprowadzę Cię teraz przez wszystkie etapy.

Najpierw musisz uruchomić generator:

```
1 $ rails generate devise:install
2
```

W oknie terminala pojawi się sporo tekstu. Jeżeli się mu przyjrzysz, zauważysz, że zawiera opis pięciu kroków, które musisz wykonać, aby zainstalować gem *Devise*. Nie obawiaj się, nie jest tak źle. Zrobimy to teraz.

Poniżej znajdują się instrukcje wzięte wprost z tekstu wyświetlonego w terminalu:

1. Sprawdź, czy zdefiniowałeś domyślne opcje adresu URL w plikach konfiguracyjnych. Poniżej przedstawiona jest przykładowa opcja środowiska programistycznego `default_url_options` w pliku *config/environments/development.rb*:

   ```
   config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
   ```

   W środowisku produkcyjnym opcja `host:` musi wskazywać serwer Twojej aplikacji.
2. Sprawdź, czy w pliku *config/routes.rb* ustawiony jest adres URL strony głównej, na przykład: `root to: "home#index"`.
3. Sprawdź, czy w pliku *app/views/layouts/application.html.erb* zdefiniowane są komunikaty flash, na przykład:

   ```
   <p class="notice"><%= notice %></p>
   <p class="alert"><%= alert %></p>
   ```
4. Jeżeli udostępniasz aplikację w usłudze Heroku tylko z wersją Rails 3.2, musisz ustawić opcję:

   ```
   config.assets.initialize_on_precompile = false
   ```
5. Skopiuj widoki gema *Devise* do swojej aplikacji (w celu ich dostosowania) za pomocą polecenia:

   ```
   rails g devise:views
   ```

Przyjrzyj się powyższym krokom. Od razu możesz pominąć krok czwarty, ponieważ nie korzystasz z wersji Rails 3.2, tylko z 4.2.1 lub nowszej.

Możesz również pominąć krok drugi, ponieważ zdefiniowałeś już główną stronę *index*.

Pozostały więc kroki 1., 3. i 5. Wykonaj je teraz.

KROK PIERWSZY

Przyjrzyj się krokowi pierwszemu. Musisz w dwóch plikach dodać po jednym wierszu kodu.

/config/environments/development.rb:

```
1 .
2 .
3   config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
4 end
5
```

Wpisz po prostu wiersz numer 3 na końcu wskazanego pliku (powyżej końcowego wiersza `end`, który jest tu pokazany).

Następnie podobną zmianę wprowadź w innym pliku:

/config/environments/production.rb:

```
1 .
2 .
3   config.action_mailer.default_url_options = { host: 'obrazkowo.herokuapp.com' }
4 end
5
```

Przyjrzyj się 3. wierszowi. Jest on bardzo podobny do 3. wiersza z poprzedniego listingu, ale zamiast opcji `host: 'localhost', port: 3000` zawiera rzeczywisty adres URL aplikacji w usłudze Heroku. W tym miejscu wpisz adres, który nadałeś swojej aplikacji.

O co tu chodzi? Otóż te dwa pliki zawierają ustawienia środowiska programistycznego i produkcyjnego. Wskazałeś w nich, aby aplikacja w lokalnym środowisku programistycznym wykorzystywała serwer WEBrick, a w produkcyjnym — usługę Heroku.

KROK TRZECI

Teraz przyjrzyj się krokowi trzeciemu.

W tym kroku musisz wpisać nieco kodu obsługującego komunikaty flash. Są to niewielkie automatyczne podpowiedzi pojawiające się na stronie, gdy użytkownik wykona jakąś operację.

Na przykład gdy użytkownik się zaloguje, wtedy u góry strony pojawi się komunikat flash typu „Pomyślnie zalogowałeś się do serwisu”. Po wylogowaniu pojawi się inny komunikat: „Pomyślnie wylogowałeś się z serwisu” itd.

Gem *Devise* wyświetla powyższe komunikaty automatycznie, ale musisz wpisać trochę kodu, aby wskazać gemowi miejsce na stronie, w którym komunikaty mają się pojawiać.

Instrukcja zawiera proponowany kod, który trzeba umieścić w pliku strony:

```
1 <p class="notice"><%= notice %></p>
2 <p class="alert"><%= alert %></p>
3
```

Trzeba go jednak nieco zmodyfikować i zmienić wygląd tych komunikatów za pomocą platformy Bootstrap. W tym celu otwórz stronę *http://getbootstrap.com/*, kliknij odnośnik *Components*, przewiń zawartość strony i kliknij odnośnik *Alerts* (alerty).

Komunikat flash jest w rzeczywistości alertem, więc użyj klasy `alert` platformy Bootstrap. Zauważ, że dostępne są cztery kolory alertów: zielony (`success`, czyli powodzenie), niebieski (`info`, informacja), żółty (`warning`, ostrzeżenie) i czerwony (`danger`, niebezpieczeństwo). Użyj niebieskiego komunikatu.

Gdzie trzeba umieścić kod, aby gem *Devise* mógł wykorzystać komunikaty flash? Muszą one pojawiać się na każdej stronie Twojej witryny, więc odpowiednim miejscem umieszczenia kodu będzie wiersz w pliku */app/views/layouts/application.html.erb*, tuż powyżej znacznika `<%= yield %>` i poniżej początku sekcji `div`:

/app/views/layouts/application.html.erb

```
1 .
2 .
3 <div class="container">
4   <% flash.each do |name, msg| %>
5     <%= content_tag(:div, msg, class: "alert alert-info") %>
6   <% end %>
7 <%= yield %>
8 </div>
9 .
10
```

Przyjrzyj się powyższemu kodowi, ponieważ wygląda nieco inaczej niż zalecany w instrukcji do gema *Devise*.

Kod zawiera typową pętlę języka Ruby, która mówi: „Z każdym komunikatem flash zrób to i to”. W tym przypadku „to i to” oznacza umieszczenie komunikatu w sekcji `div` z przypisaną klasą `alert alert-info`.

Skąd wzięła się klasa `alert alert-info`? Skopiowałeś ją ze strony *GetBootStrap.com* z fragmentu kodu w sekcji *Alerts*. Ten komunikat będzie niebieski.

Zwróć również uwagę, że znacznik otwierający w 1. wierszu `<% flash.each do |name, msg| %>` nie zawiera znaku równości (zawiera go wiersz 5., ale nie 4.)!

Co to oznacza? Do tej pory wszystkie osadzone obiekty Ruby zawierały znacznik otwierający <%=. Znak równości oznacza „wyświetl tę treść na ekranie". Jednak nie chcemy, aby pętla wyświetlała cokolwiek na ekranie. Treść ma się pojawiać tylko wtedy, gdy warunek wewnątrz pętli będzie spełniony. W wierszu 5. znajduje się znacznik <%=, ponieważ jego zawartość ma być wyświetlana na ekranie.

KROK PIĄTY

Ostatnią czynnością do wykonania jest wpisanie w terminalu polecenia podanego w instrukcji. Zrób to więc teraz:

```
1 $ rails g devise:views
2
```

Polecenie to uruchamia generator widoków gema *Devise*. Zwróć uwagę, że wcześniej używałeś już polecenia `generate`, na przykład:

```
1 $ rails generate devise:views
2
```

Pojedyncza litera `g` zamiast `generate` jest skrótem, który można bez przeszkód stosować.

Po wykonaniu powyższego polecenia sprawdź zawartość katalogu */app/views/*. Powinien pojawić się w nim nowy katalog *devise*. Zawiera on kilka innych katalogów ze wszystkimi stronami, które gem *Devise* utworzył na potrzeby obsługi takich operacji jak logowanie i wylogowywanie użytkowników, zmiana profilu, resetowanie hasła itp. Sprawdź to!

Również w terminalu powinno pojawić się sporo tekstu. Jest to lista stron utworzonych przez gem *Devise*. Przejrzyj tę listę. Czy widzisz w niej nazwy stron? Przyjrzyj się na przykład plikowi */app/views/devise/registrations/new.html.erb*.

Zapewne się domyślasz, że jest to strona umożliwiająca użytkownikowi zarejestrowanie się w Twojej aplikacji (tworząca „nową" rejestrację). Otwórz ten plik w edytorze. Na samym początku zobaczysz tytuł *Sign Up* (zarejestruj się). Usuń te słowa i w ich miejsce wpisz **`Rejestracja`**.

> **Uwaga!** Gem *Devise* domyślnie tworzy strony z napisami w języku angielskim. Aby na stronach pojawiały się polskie napisy, należy zmienić pliki utworzone przez gem. Szczegółowe informacje, jak to zrobić, zawarte są w dodatku, który możesz pobrać pod adresem *ftp://ftp.helion.pl/przyklady/rrtwww.zip*.

A co to jest za plik *app/views/devise/sessions/new.html.erb*? Jest to strona do logowania użytkowników. Gdy użytkownik się zaloguje, tworzona jest sesja. Operacja logowania polega na tworzeniu nowej sesji.

Analogicznie wylogowanie oznacza zamknięcie sesji.

Gem *Devise* utworzył naprawdę dużo plików i prawdę mówiąc, z większości z nich nie będziesz korzystał. Nie czuj się więc przytłoczony.

Instalacja gema jest *prawie zakończona*, pozostała jeszcze jedna czynność do wykonania. Gem został skonfigurowany, widoki utworzone, ale nie ma połączenia z bazą danych, w której będą przechowywane informacje o użytkownikach. Na szczęście nie jest to problem, wystarczy wpisać poniższe dwa polecenia:

```
1 $ rails generate devise user
2 $ rake db:migrate
3
```

OBSŁUGA BAZY DANYCH W PLATFORMIE RAILS

Tak naprawdę to nie pisałem jeszcze o bazach danych, więc zrobię to teraz. Dzięki platformie Rails praca z bazami jest wyjątkowo prosta, ale trzeba wiedzieć, co się z nimi dzieje.

Za każdym razem gdy będziesz chciał coś zrobić z bazą danych za pomocą platformy Rails, będziesz musiał utworzyć tzw. **migrację** i wysłać ją do bazy. Migrację możesz traktować jak listę rzeczy do zrobienia. Utwórz listę, a następnie przekaż ją do bazy danych.

Przed chwilą, wpisując polecenie `rails generate devise user`, utworzyłeś migrację i „dodałeś" użytkowników. Możesz nawet zobaczyć zawartość pliku z migracją. Pliki z migracjami znajdują się w katalogu */db/migrate*.

W moim przypadku polecenie `rails generate devise user` utworzyło następujący plik z migracją:

/db/migrate/20150824184627_devise_create_users

```
 1 class DeviseCreateUsers < ActiveRecord::Migration
 2  def change
 3    create_table(:users) do |t|
 4      ## Database authenticatable
 5        t.string :email,              null: false, default: ""
 6        t.string :encrypted_password, null: false, default: ""
 7
 8        ## Recoverable
 9        t.string   :reset_password_token
```

```
10        t.datetime :reset_password_sent_at
11
12       ## Rememberable
13       t.datetime :remember_created_at
14
15       ## Trackable
16       t.integer  :sign_in_count, default: 0, null: false
17       t.datetime :current_sign_in_at
18       t.datetime :last_sign_in_at
19       t.string   :current_sign_in_ip
20       t.string   :last_sign_in_ip
21
22       ## Confirmable
23       # t.string   :confirmation_token
24       # t.datetime :confirmed_at
25       # t.datetime :confirmation_sent_at
26       # t.string   :unconfirmed_email # Only if using reconfirmable
27
28       ## Lockable
29       # t.integer  :failed_attempts, default: 0, null: false # Only if lock
           ↪strategy is :failed_attempts
30       # t.string   :unlock_token # Only if unlock strategy is :email or :both
31       # t.datetime :locked_at
32
33       t.timestamps null: false
34     end
35
36     add_index :users, :email,                unique: true
37     add_index :users, :reset_password_token, unique: true
38     # add_index :users, :confirmation_token,   unique: true
39     # add_index :users, :unlock_token,         unique: true
40   end
41 end
```

Jest tu sporo kodu, ale tak naprawdę nie musisz wiedzieć, co on realizuje. Przejrzyj go pobieżnie i sprawdź, czy coś Ci się nie rzuciło w oczy.

Zwróć uwagę, że kod zawiera odwołanie do adresu e-mail i hasła (ponieważ użytkownicy będą logowali się za pomocą tych danych). Rejestruje również czas zalogowania się użytkownika (wiersz 17.) i resetuje hasło (wiersz 38.).

Jak wcześniej wspomniałem, w tej chwili (a nawet w ogóle) nie musisz wiedzieć, co oznacza każdy wiersz powyższego kodu. Pamiętaj tylko, że utworzyłeś migrację.

Do wysłania migracji do bazy danych służy polecenie `rake db:migrate`.

BAZA PROGRAMISTYCZNA I BAZA PRODUKCYJNA

Zanim przejdziemy dalej, muszę nieco powiedzieć o bazach danych (ponieważ już ich używasz). Pierwszy raz wspomniałem o nich przy okazji omawiania usługi Heroku i teraz chciałbym wrócić do tego tematu.

W środowisku programistycznym wykorzystywana jest baza danych o nazwie ***sqlite3***, bardzo prosta, instalowana razem z platformą Rails. Gdy przyjrzysz się plikowi *Gemfile*, znajdziesz w nim referencję do tej bazy.

Jak wspomniałem wcześniej, sqlite3 nie nadaje się do obsługi produkcyjnej witryny. Do tego celu trzeba użyć innej bazy.

Użyjesz bazy PostgreSQL. Dlaczego właśnie tej? Ponieważ usługa Heroku lubi tę bazę, która w dodatku jest bardzo prosta w obsłudze. Jedyną operacją, którą trzeba wykonać, jest dodanie gema *postgres* (co zrobiłeś już wcześniej). Jak pamiętasz, była to czynność nieco bardziej złożona niż dodanie zwykłego gema, ponieważ trzeba było wskazać, że aplikacja w środowisku programistycznym powinna wykorzystywać bazę sqlite3, a w produkcyjnym — PostgreSQL. Ale i tak było to dość proste.

Oto co wtedy zrobiłeś:

/Gemfile

```
 1 source 'https://rubygems.org'
 2 gem 'rails', '4.2.1'
 3 gem 'sass-rails', '~> 5.0'
 4 gem 'uglifier', '>= 1.3.0'
 5 gem 'coffee-rails', '~> 4.1.0'
 6 gem 'jquery-rails'
 7 gem 'turbolinks'
 8 gem 'jbuilder', '~> 2.0'
 9 gem 'sdoc', '~> 0.4.0', group: :doc
10 gem 'bootstrap-sass', '~> 3.3.5.1'
11 gem 'devise', '~> 3.5.2'
12
13 group :development, :test do
14  gem 'byebug'
15   gem 'web-console', '~> 2.0'
16   gem 'spring'
17 end
18
19 group :development, :test do
20   gem 'sqlite3'
21 end
22
23 group :production do
```

```
24   gem 'pg', '0.18.2'
25   gem 'rails_12factor', '0.0.3'
26 end
27
```

W powyższym pliku wprowadziłeś dwie zmiany. Przede wszystkim usunąłeś oryginalne odwołanie do gema *sqlite3*, umieszczone w chwili utworzenia projektu. Odwołanie to jest jednak potrzebne, więc umieściłeś je w grupie odwołań w wierszu 19.

Grupa ta zawiera informację, że Twoja aplikacja będzie korzystać z bazy sqlite3 w środowisku programistycznym i testowym (pamiętaj, że książka nie zawiera opisu żadnych operacji wykonywanych w środowisku testowym).

Dodatkowo umieściłeś odwołania do gemów baz danych pg i rails_12factor (baza wymagana w usłudze Heroku), które będą wykorzystywane w środowisku produkcyjnym.

Za każdym razem, gdy dodasz gem do pliku *Gemfile*, musisz użyć polecenia `bundle install`, jednak tym razem będzie nieco inaczej. To polecenie nie może obejmować opcji środowiska produkcyjnego. To logiczne, ponieważ w środowisku programistycznym nie powinna być instalowana baza PostgreSQL. Poniżej przedstawione jest odpowiednie polecenie:

```
1 $ bundle install --without production
2
```

> **Uwaga!** Przed słowem `without` znajdują się dwa myślniki `--`.

Powyższe polecenie trzeba wprowadzić tylko raz. Platforma Rails będzie od tej chwili pamiętać, aby nie instalować gemów przeznaczonych dla środowiska produkcyjnego.

Dlaczego znów o tym piszę? Ponieważ chcę mieć pewność, iż wiesz, że będziesz korzystał z dwóch różnych baz danych. Musisz o tym koniecznie pamiętać. Dlaczego? Ponieważ za każdym razem po utworzeniu migracji trzeba ją wysyłać do lokalnej, a nie do produkcyjnej bazy danych. Natomiast w usłudze Heroku trzeba migrację wysłać do bazy produkcyjnej (w tym przypadku PostgreSQL).

WYSYŁANIE MIGRACJI DO BAZY POSTGRESQL W USŁUDZE HEROKU

Gdy użyłeś polecenia `rails generate devise user`, utworzyłeś migrację z obsługą użytkowników. Następnie za pomocą polecenia `rake db:migrate` wysłałeś migrację do bazy danych, konkretnie do bazy sqlite3.

Nie wysłałeś migracji do bazy PostgreSQL w usłudze Heroku, ale musisz to zrobić. Za każdym razem gdy będziesz wysyłał migrację do lokalnej bazy danych w środowisku programistycznym (sqlite3), będziesz musiał również wysyłać ją do bazy w środowisku produkcyjnym (PostgreSQL). Służy do tego następujące polecenie:

```
1 $ heroku run rake db:migrate
2
```

Polecenie wygląda niemal identycznie jak `rake db:migrate` użyte lokalnie, zawiera jedynie na początku słowa `heroku run`. Pamiętaj, że jeżeli nie użyjesz tego polecenia, baza danych w usłudze Heroku nie będzie działała prawidłowo.

SPRAWDZENIE NOWYCH STRON GEMA DEVISE

Idźmy dalej! Zakończyłeś instalację gema *Devise*, wysłałeś migrację do bazy, dzięki czemu gem może obsługiwać użytkowników zarówno w środowisku programistycznym, jak i produkcyjnym, oraz utworzyłeś strony umożliwiające użytkownikom logowanie i wylogowywanie z aplikacji.

Czas więc zobaczyć te strony!

Przede wszystkim zatrzymaj i ponownie uruchom serwer.

Następnie aby sprawdzić, gdzie te strony się znajdują, odszukaj ścieżki utworzone przez gem *Devise*. W tym celu użyj znanego Ci już polecenia `rake routes`:

```
1 $ rake routes
2
```

Jak pamiętasz, gdy poprzednim razem użyłeś tego polecenia, w aplikacji istniały tylko dwie ścieżki (do stron *index* i *O mnie*). Tym razem jest ich o wiele więcej!

```
1 $ rake routes
2                   Prefix Verb   URI Pattern                    Controller#Action
3         new_user_session GET    /users/sign_in(.:format)       devise/sessions#new
4             user_session POST   /users/sign_in(.:format)       devise/sessions#create
5 .
6 .
```

Jak poprzednio, pierwsza kolumna zawiera ścieżkę. W drugiej znajduje się informacja, czy jest to zwykła strona (komenda `GET`), czy formularz (`POST`), czy strona do wylogowania użytkownika (`DELETE`). W trzeciej kolumnie znajduje się adres URL, którego użyjesz do wyświetlenia żądanej strony w przeglądarce. Piąta kolumna zawiera opis operacji kontrolera.

Powyżej nie jest przedstawiona pełna lista ścieżek, tylko jej początek, ale możesz wprowadzić powyższe polecenie i zobaczyć pełną listę.

Pamiętaj, że ścieżki zawierające komendę `GET` oznaczają zwykłe strony WWW. Wybierz zatem jedną z nich, wpisz adres w przeglądarce i zobacz, jak strona wygląda!

Wpisz następujący adres: **`https://obrazkowo-programista.c9.io/users/sign_up`** (lub inny własny adres URL).

(Strona gema Devise do rejestrowania użytkowników *https://obrazkowo-programista.c9.io/users/sign_up*)

Sprawdź również stronę logowania — **`https://obrazkowo-programista.c9.io/users/sign_in`** (lub inny własny adres URL):

(Strona gema Devise do logowania użytkowników *https://obrazkowo-programista.c9.io/users/sign_in*)

Na tej stronie znajduje się odnośnik dla użytkownika, którzy zapomniał, jak brzmi jego hasło, i musi je zresetować. Tę stronę też możesz sprawdzić.

Utworzone strony i formularze są bardzo funkcjonalne. Za ich pomocą użytkownik może założyć konto, zalogować się itd. Oczywiście strony te na razie wyglądają dość skromnie, ale sama funkcjonalność działa prawidłowo. Teraz nadasz stronom elegancki wygląd.

ZMIANA WYGLĄDU STRON GEMA DEVISE

Teraz zajmij się poprawieniem wyglądu stron. To całkiem proste. Wszystkie pliki, które trzeba zmienić, znajdują się w katalogu */app/views/devise*. Zacznij od strony służącej do rejestrowania użytkownika:

/app/views/devise/registrations/new.html.erb

```
<h2>Rejestracja</h2>

<%= form_for(resource, as: resource_name, url: registration_path(resource_name))
➥do |f| %>
  <%= devise_error_messages! %>

  <div class="field">
    <%= f.label :email %><br />
    <%= f.email_field :email, autofocus: true %>
  </div>

  <div class="field">
    <%= f.label :hasło %>
    <% if @minimum_password_length %>
    <em>(minimum <%= @minimum_password_length %> znaków)</em>
    <% end %><br />
    <%= f.password_field :password, autocomplete: "off" %>
  </div>

  <div class="field">
    <%= f.label :potwierdź_hasło %><br />
    <%= f.password_field :password_confirmation, autocomplete: "off" %>
  </div>

  <div class="actions">
    <%= f.submit "Wyślij" %>
  </div>
<% end %>

<%= render "devise/shared/links" %>

```

Ten plik może wyglądać na najbardziej skomplikowany z tych, jakie do tej pory widziałeś, ale nie jest tak źle. Aby zmienić wygląd strony gema *Devise*, użyjesz oczywiście platformy Bootstrap!

Otwórz stronę *GetBootstrap.com* i kliknij odnośnik *CSS* u góry ekranu. Następnie kliknij *Forms* (formularze) po prawej stronie. Teraz musisz zrobić kilka rzeczy. Nie będziesz kopiował i wklejał całego kodu widocznego pod przykładowym formularzem. Zamiast tego wybierzesz tylko jego dwa fragmenty.

Po pierwsze, platforma Bootstrap wymaga, aby każde pole i etykieta formularza były umieszczone w sekcji `div` z klasą `form-group`:

```
1 <div class="form-group">
2 .
3 .
4 </div>
```

W pliku *new.html.erb* elementy te już są umieszczone w sekcji `div`, ale jej klasa nosi nazwę `field`. Możesz to sprawdzić w wierszach 6., 11. i 19. na powyższym listingu.

To żaden problem. Po prostu w wierszach 6., 11. i 19. zmień słowo `field` na `form-group`.

Po drugie, każdemu polu formularza trzeba przypisać klasę `form-control` (pole formularza to miejsce, w którym wpisuje się dane). W pliku *new.html.erb* pola te są umieszczone w wierszach 8., 16. i 21.:

```
 7
 8 <%= f.email_field :email, autofocus: true, class: 'form-control' %>
 9

15
16 <%= f.password_field :password, autocomplete: "off", class: 'form-control' %>
17

20
21 <%= f.password_field :password_confirmation, autocomplete: "off",
   ↪class: 'form-control' %>
22
```

Po trzecie, zwróć uwagę, że formularz wymaga, aby użytkownik wpisał hasło, a następnie je potwierdził. Według mnie nie ma potrzeby, aby w tak prostej aplikacji użytkownik dwukrotnie wpisywał hasło, więc można się pozbyć dodatkowego pola. Wystarczy w tym celu usunąć wiersze od 19. do 22.

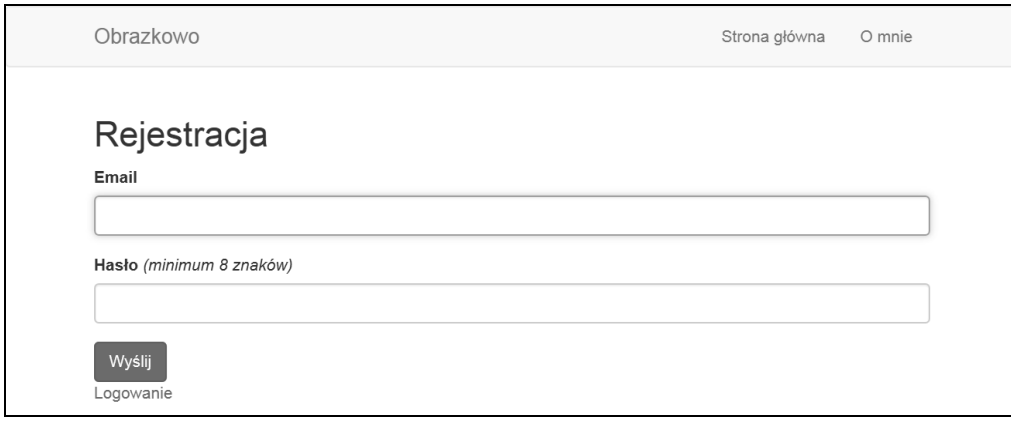
Po czwarte, co zrobić z przyciskiem *Wyślij*? Wygląda dość ubogo… Można tu użyć komponentu `button` platformy Bootstrap w taki sam sposób jak na stronie *index*. Wpisz następujący kod:

```
19
20 <%= f.submit "Wyślij", class: 'btn btn-primary' %>
21
```

Zmieniony plik wygląda teraz tak:

/app/views/devise/registrations/new.html.erb

```
 1 <h2>Rejestracja</h2>
 2
 3 <%= form_for(resource, as: resource_name, url: registration_path(resource_name))
   ↪do |f| %>
 4   <%= devise_error_messages! %>
 5
 6   <div class="form-group">
 7     <%= f.label :email %><br />
 8     <%= f.email_field :email, autofocus: true, class: 'form-control' %>
 9   </div>
10
11   <div class="form-group">
12     <%= f.label :hasło %>
13     <% if @minimum_password_length %>
14     <em>(minimum <%= @minimum_password_length %> znaków)</em>
15     <% end %><br />
16     <%= f.password_field :password, autocomplete: "off", class: 'form-control' %>
17   </div>
18
19   <div class="actions">
20     <%= f.submit "Wyślij", class: 'btn btn-primary' %>
21   </div>
22 <% end %>
23
24 <%= render "devise/shared/links" %>
25
```



(Strona gema Devise do rejestrowania użytkowników *https://obrazkowo-programista.c9.io/users/sign_up*)

Teraz strona wygląda zdecydowanie lepiej, ale dodając panele, możesz nadać jej jeszcze lepszy wygląd. Wróć do strony *GetBootstrap.com*, kliknij odnośniki *Components*, a następnie *Panels* (panele) po prawej stronie ekranu.

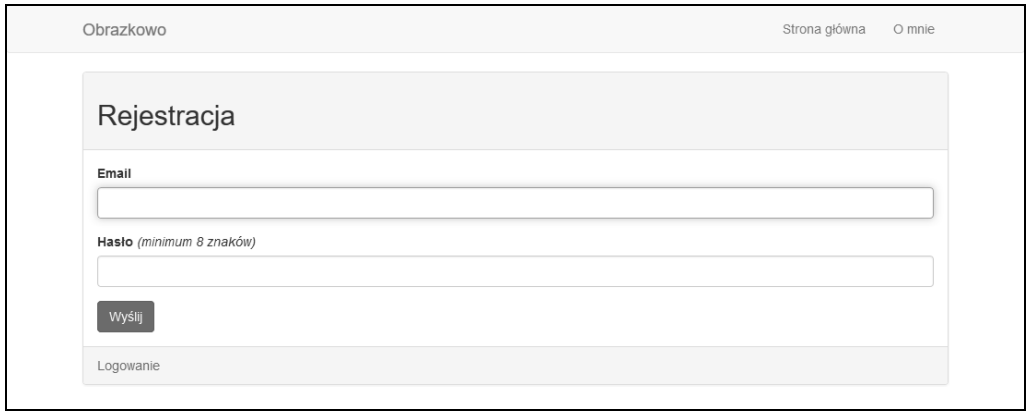
Dodasz teraz komponenty *Panel with heading* (panel z nagłówkiem) oraz *Panel with footing* (panel ze stopką) u dołu swojej strony w miejscu, gdzie znajduje się odnośnik *Logowanie*. Poniższy kod pokazuje, jak to zrobić:

/app/views/devise/registrations/new.html.erb

```
<div class="panel panel-default">
 <div class="panel-heading"><h2>Rejestracja</h2></div>
    <div class="panel-body">

<%= form_for(resource, as: resource_name, url: registration_path(resource_name))
↪do |f| %>
  <%= devise_error_messages! %>

  <div class="form-group">
    <%= f.label :email %><br />
    <%= f.email_field :email, autofocus: true, class: 'form-control' %>
  </div>

  <div class="form-group">
    <%= f.label :hasło %>
    <% if @minimum_password_length %>
    <em>(minimum <%= @minimum_password_length %> znaków)</em>
    <% end %><br />
    <%= f.password_field :password, autocomplete: "off", class: 'form-control' %>
  </div>

  <div class="actions">
    <%= f.submit "Wyślij", class: 'btn btn-primary' %>
  </div>
<% end %>

    </div>
  <div class="panel-footer"><%= render "devise/shared/links" %></div>
</div>

```

Zmiany wyróżnione są pogrubioną czcionką. Zawartość całej strony została umieszczona wewnątrz sekcji `div` z opcją `class="panel panel-default"`. Nagłówek strony został umieszczony wewnątrz sekcji `div` z opcją `class="panel-heading"`, główna treść strony — w sekcji `div` z opcją `class="panel-body"`, a odnośniki — w sekcji `div` z opcją `class="panel-footer"`.

Wynik zmian powinien wyglądać jak niżej:



(Strona gema Devise do rejestrowania użytkowników
https://obrazkowo-programista.c9.io/users/sign_up o poprawionym wyglądzie)

Teraz możesz samodzielnie wprowadzić te same zmiany w innych formularzach gema *Devise*:

/app/views/devise/registrations/edit.html.erb (strona do zmiany profilu użytkownika)

/app/views/devise/sessions/new.html.erb (strona do logowania użytkownika)

/app/views/devise/passwords/new.html.erb (strona do resetowania zapomnianego hasła)

Zauważ, że powyższe strony wyglądają dość podobnie, więc łatwo możesz wprowadzić w nich te same zmiany co w pliku */app/views/devise/registrations/new.html.erb*.

TWORZENIE ODNOŚNIKÓW NA STRONACH GEMA DEVISE

Teraz, gdy strony utworzone przez gem *Devise* wyglądają całkiem dobrze, trzeba w pasku nawigacyjnym umieścić odnośniki. Można to zrobić, wykorzystując znacznik `link_to`, tak jak poprzednio, ale tym razem wprowadzimy małą odmianę.

Odnośniki nie będą widoczne na wszystkich stronach dla wszystkich odwiedzających je użytkowników. Na przykład użytkownik nie będzie widział odnośnika *Edycja profilu*, jeżeli nie będzie zalogowany (a tym bardziej jeżeli się nie zarejestruje!).

Ponadto użytkownik, który już się zalogował, nie powinien widzieć odnośnika do strony logowania.

Potrzebny jest również odnośnik do wylogowania, który wygląda nieco inaczej niż odnośniki utworzone wcześniej.

SPRAWDZENIE, CZY UŻYTKOWNIK JEST ZALOGOWANY

Najpierw zajmijmy się najważniejszymi rzeczami, czyli w jaki sposób sprawdzić, czy dany użytkownik jest zalogowany, czy nie. Na szczęście dzięki platformie Rails jest to bardzo proste, trzeba w tym celu wpisać nieco kodu Ruby z prostą instrukcją `if`.

Kod ten można przetłumaczyć następująco: „Jeżeli użytkownik jest zalogowany, wyświetl podane odnośniki, a w przeciwnym wypadku wyświetl inne". Poniżej przedstawiony jest ten kod:

```
1 <% if user_signed_in? %>
2 .
3 .
4 <% else %>
5 .
6 .
7 <% end %>
8
```

Odnośniki, które powinny być widoczne tylko wtedy, gdy użytkownik będzie zalogowany (edycja profilu, wylogowanie), trzeba umieścić pomiędzy pierwszym wierszem a instrukcją `else`. Odnośniki, które będą wyświetlane, gdy użytkownik nie będzie zalogowany (logowanie, rejestracja), zostaną umieszczone pomiędzy instrukcjami `else` i `end`.

Aby poznać ścieżki prowadzące do odpowiednich stron (edycja profilu, logowanie, wylogowanie, rejestracja), użyj dobrze znanego Ci polecenia `rake routes`:

```
1 $ rake routes
2
```

Zwróć szczególną uwagę na ścieżki zawierające komendę `GET` (pamiętaj o dodaniu końcówki `_path` do zmiennych w poleceniu `link_to`):

Logowanie: `new_user_session_path`

Edycja profilu: `edit_user_registration_path`

Rejestracja: `new_user_registration_path`

Wylogowanie: `destroy_user_session_path`

Przyjrzyjmy się ostatniej ścieżce, oznaczającej stronę wylogowania. Wynik polecenia `rake routes` pokazuje, że ścieżka ta zawiera komendę `DELETE` (usuń), a nie `GET` (pobierz). To zrozumiałe, ponieważ wylogowanie z aplikacji oznacza „usunięcie" sesji.

Jednak odnośnik do komendy `DELETE` tworzy się nieco inaczej niż zwykły odnośnik. Wygląda on następująco:

```
1 <%= link_to 'Wylogowanie', destroy_user_session_path, method: :delete %>
2
```

Zwróć uwagę, że kod wygląda podobnie jak w przypadku zwykłego odnośnika, tylko na końcu dodana jest opcja `method: :delete`. To bardzo proste.

ZMIANA PASKA NAWIGACYJNEGO

Masz zatem przygotowaną instrukcję `if`, jak również odnośniki do stron gema *Devise*, które zamierzasz dodać. Zmień pasek nawigacyjny:

/app/views/home/_header.html.erb

```
 1 <nav class="navbar navbar-default" role="navigation">
 2   <div class="container">
 3     <!-- Brand and toggle get grouped for better mobile display -->
 4     <div class="navbar-header">
 5       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
 6         data-target="#bs-example-navbar-collapse-1">
 7       <span class="sr-only">Toggle navigation</span>
 8         <span class="icon-bar"></span>
 9         <span class="icon-bar"></span>
10         <span class="icon-bar"></span>
11       </button>
12       <%= link_to 'Obrazkowo', root_path, class: 'navbar-brand' %>
13     </div>
14
15     <!-- Collect the nav links, forms, and other content for toggling -->
16     <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
17       <ul class="nav navbar-nav navbar-right">
18         <li><%= link_to 'Strona główna', root_path %></li>
19         <li><%= link_to 'O mnie', home_about_path %></li>
20
21         <% if user_signed_in? %>
22           <li><%= link_to 'Edycja profilu', edit_user_registration_path %></li>
23           <li><%= link_to 'Wylogowanie', destroy_user_session_path,
             ↪method: :delete %></li>
24         <% else %>
25           <li><%= link_to "Logowanie", new_user_session_path %></li>
26           <li><%= link_to "Rejestracja", new_user_registration_path %></li>
27         <% end %>
28
29       </ul>
30     </div><!-- /.navbar-collapse -->
31   </div><!-- /.container-fluid -->
32 </nav>
33
```

Zmiana polega na dodaniu wierszy od 21. do 27. Zwróć uwagę, że instrukcja `if` sprawdzająca, czy użytkownik jest zalogowany, została umieszczona poniżej odnośników *Strona główna* i *O mnie*, ponieważ odnośniki te muszą być wyświetlane niezależnie od tego, czy użytkownik jest zalogowany, czy nie.

Zrobiłeś już całkiem sporo. Użytkownicy mogą się rejestrować, logować, zmieniać swoje profile, a pasek nawigacyjny zmienia się dynamicznie w zależności od tego, czy użytkownik jest zalogowany, czy nie.

Teraz pora na dodanie do aplikacji prawdziwej funkcjonalności umożliwiającej użytkownikom ładowanie „pinów" na stronę. Zajmiemy się tym w następnym rozdziale.

SKOROWIDZ

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

RUBY ON RAILS — PROSTY, WYDAJNY, CIEKAWY!

Ruby on Rails jest otwartą platformą programistyczną umożliwiającą pisanie stron WWW prosto, wydajnie i… bardzo przyjemnie. Wykorzystuje architekturę MVC (model – widok – kontroler), co skraca okres żmudnego kodowania. Dzięki temu programista może skupić się na pracy twórczej. Narzędzie to wykorzystują takie serwisy jak Groupon, Indiegogo, Airbnb, Yammer, SoundCloud, Scribd, Shopify, Hulu i wiele innych. Krążą opinie, że praca z Ruby on Rails jest pasmem frustracji — nic bardziej mylnego! Z tą książką sprawnie przygotujesz sobie wygodne środowisko pracy, niezależnie od tego, z jakiego systemu operacyjnego korzystasz na co dzień. Dowiesz się, jak projektować i tworzyć rozbudowane serwisy internetowe, a Twoja nauka będzie polegać na tworzeniu rzeczywistej, działającej aplikacji! Nawet jeśli dopiero piszesz pierwsze linie kodu, wkrótce zaczniesz tworzyć rozbudowane projekty. Autor przystępnie i interesująco wyjaśnia wszystkie kwestie, które trzeba znać, aby tworzyć nowoczesne aplikacje — od tych najbardziej podstawowych aż po oparte na zaawansowanych ideach, takich jak korzystanie z gemów czy modelu MVC.

W tej książce autor przedstawił między innymi następujące zagadnienia:

- przygotowanie i uruchomienie środowiska pracy oraz kontrola wersji aplikacji (systemy GitHub i Bitbucket)
- korzystanie z komponentów Bootstrap
- obsługa baz danych za pomocą platformy Rails
- uwierzytelnianie użytkowników i ich obsługa
- wykorzystanie gemu paperclip do ładowania obrazów
- stylizacja i upiększanie witryny za pomocą jQuery i pinów

John Elder

Mieszka w Chicago. Ten programista weteran ze znanego serwisu Codemy.com zajmuje się programowaniem od siódmego roku życia. Zbudował jedną z pierwszych internetowych sieci reklamowych. Rozwijał program Submission--Spider — jedno z pierwszych narzędzi do optymalizacji wyszukiwarek internetowych, znane milionom użytkowników w dwudziestu kilku krajach. Dzisiaj jest uznanym autorytetem w dziedzinie aplikacji internetowych i programowania, jest też znawcą tematyki ataków sieciowych i reklam w internecie. Ma przy tym cenną umiejętność przekazywania wiedzy w sposób przystępny, interesujący i zrozumiały.

Helion
40531 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-1843-4
9 788328 318434
cena: 32,90 zł