KONSPEKT WARSZTATÓW RUBY ON RAILS.

TEMAT: ZAPOZNANIE SIĘ Z TEMATYKĄ RUBY ON RAILS NA PRZYKŁADZIE BLOGA.

Spis treści

1. Wprowadzenie

Źródła aplikacji są publiczne i dostępne pod adresem:
`https://github.com/railwaymen/workshops_agh_blog`

Pobranie repozytorium na dysk:

```
$ git clone git://github.com/railwaymen/workshops_agh_blog.git
```

Przejście pomiędzy kolejnymi krokami:

```
$ ./goto 0.x
```

2. Panel administratora do zarządzania postami

W pliku `Gemfile` należy dodać linijkę:

```
gem 'nifty-generators', group: :development
```

Następnie w konsoli wykonujemy polecenia:

```
$ bundle
$ rails g nifty:layout
$ mv public/stylesheets/application.css app/assets/stylesheets/styles.css
```

W pliku layoutu `app/views/layouts/application.html.erb` należy poprawić linijkę odpowiedzialną za dołączanie javascriptów:

```
6 <%= javascript_include_tag "application" %>
```

Proponuję też zmienić tytuł strony:

```
4 <title>Blog</title>
```

Powracamy do konsoli i wykonujemy polecenia:

```
$ rails g nifty:scaffold Admin::Post title:string content:text
$ bundle
$ rake db:migrate
$ rm public/index.html
```

W drugiej konsoli uruchamiamy serwer aplikacji:

```
$ rails s
```

Pod adresem `http://localhost:3000/admin/posts` możemy sprawdzić działanie aplikacji. Operacje na postach są już w pełni funkcjonalne. Możemy dodawać nowe posty, oglądać i edytować istniejące oraz listować wszystkie.

3. Wyświetlenie postów na stronie

W konsoli generujemy zasób dostępny dla wszystkich:

```
$ rails g nifty:scaffold Post index show
```

W pliku `config/routes.rb` dopisujemy linijkę:

```
6 root to: 'posts#index'
```

W widoku `app/views/posts/index.html.erb` podmieniamy treść:

```
1 <% @posts.each do |post| %>
2   <div>
3     <h3><%= link_to post.title, post %></h3>
4     <p>on <%= post.created_at.to_date.to_s(:long) %></p>
5     <p><%= post.content %></p>
6   </div>
7 <% end %>
```

4. Zabezpieczenie panelu administratora

Dodanie funkcjonalności uwierzytelniania użytkowników rozpoczynamy od wpisu w pliku `Gemfile`:

```
gem 'devise'
```

Przenosimy się do konsoli:

```
$ bundle
$ rails g devise:install
$ rails g devise User
```

Zajmijmy się teraz plikiem migracji `db/migrate/xxx_devise_create_users.rb`. Wymaga on modyfikacji, ponieważ będziemy potrzebować tylko niektórych pól w modelu User. Ostateczna treść powinna wyglądać następująco:

```
1  class DeviseCreateUsers < ActiveRecord::Migration
2    def change
3      create_table(:users) do |t|
4        t.string :email, null: false, default: ''
5        t.string :encrypted_password, null: false, default: ''
6        t.string :name
7        t.boolean :admin, default: false
8        t.timestamps
9      end
10     add_index :users, :email, unique: true
11   end
12 end
```

Model `app/models/user.rb` także wymaga poprawy:

```
1  class User < ActiveRecord::Base
2    devise :database_authenticatable, :validatable
3    attr_accessible :email, :name, :password, :password_confirmation
4    validates :email, :name, presence: true
5  end
```

Model jest dostosowany, zatem można uruchomić migrację w konsoli:

```
$ rake db:migrate
```

Powinniśmy dopasować layout. W pliku `app/views/layouts/application.html.erb` wstawiamy

```
12 <p>
13   <% if signed_in? %>
14     You are signed in as <%= current_user.name %> |
15     <%= link_to 'Sign out', destroy_user_session_path, method: :delete %>
16   <% else %>
17     <%= link_to 'Sign in', new_user_session_path %>
18   <% end %>
19 </p>
```

Na koniec pozostała kwestia autoryzacji zarządzania postami. W kontrolerze `app/controllers/admin/posts_controller.rb` stosujemy metodę before_filter:

```
2  before_filter :authenticate_user!, :authorize_user!

43 private
44
45 def authorize_user!
46   redirect_to root_path, alert: 'Access denied!' unless current_user.admin?
47 end
```

5. Dodawanie komentarzy

W konsoli generujemy zasób komentarzy:

```
$ rails g nifty:scaffold Comment content:text post:references
user:references create
$ rake db:migrate
```

W pliku `config/routes.rb` usuwamy linijkę:

```
2 resources :comments
```

oraz zmieniamy:

```
4 resources :posts, only: [:index, :show] do
5   resources :comments, only: [:create]
6 end
```

Przyszła kolej na modyfikację modeli, aby dostosować je do nowych relacji:
`app/models/comment.rb`

```
1 class Comment < ActiveRecord::Base
2   belongs_to :post
3   belongs_to :user
4   attr_accessible :content, :post_id
5   validates :content, presence: true
6 end
```

app/models/post.rb

```
2 has_many :comments
```

app/models/user.rb

```
4 has_many :comments
```

Modyfikujemy nowy kontroler `app/controllers/comments_controller.rb`:

```ruby
class CommentsController < ApplicationController
  before_filter :authenticate_user!

  def create
    @post = Post.find(params[:post_id])
    @comment = current_user.comments.build(params[:comment])
    @comment.post = @post
    if @comment.save
      redirect_to post_path(@post), notice: 'Successfully created comment.'
    else
      redirect_to post_path(@post), alert: 'Can not create comment.'
    end
  end
end
```

Kontroler `app/controllers/posts_controller.rb` także wymaga modyfikacji:

```ruby
@comments = @post.comments.all
@comment = @post.comments.build
```

Za wyświetlenie pojedynczego komentarza odpowiedzialny będzie partial: `app/views/comments/_comment.html.erb`:

```
<div>
  <p><strong><%= comment.user.name %></strong> said:<br />
  <%= comment.content %></p>
</div>
```

Pozostało jeszcze wyświetlić komentarze i formularz dodawania nowego komentarza na widoku pojedynczego posta `app/views/posts/show.html.erb`:

```
<p>
  <strong>Comments:</strong>
</p>
<div id="comments">
  <%= render @comments %>
</div>

<%= form_for [@post, @comment] do |f| %>
  <%= f.text_area :content, cols: 30, rows: 5 %>
  <div>
    <%= f.submit 'Comment' %>
```

```
  </div>
<% end %>
```

Kod dodawania komentarzy do postów jest w pełni funkcjonalny, ale możemy pokusić się o pewną zmianę: niech to dzieje się asynchronicznie przy użyciu technologii AJAX. Będzie to bardzo proste. W powyższym widoku zmieniamy linijkę:

```
<%= form_for [@post, @comment], remote: true do |f| %>
```

Kontroler `app/controllers/comments_controller.rb` wygląda teraz następująco:

```
class CommentsController < ApplicationController
  before_filter :authenticate_user!

  def create
    @post = Post.find(params[:post_id])
    @comment = current_user.comments.build(params[:comment])
    @comment.post = @post
    if @comment.save
      respond_to do |format|
        format.html { redirect_to post_path(@post), notice:
          'Successfully created comment.' }
        format.js { }
      end
    else
      respond_to do |format|
        format.html { redirect_to post_path(@post), alert: 'Can not
          create comment.' }
        format.js { render text: "alert('Can not create comment.');" }
      end
    end
  end
end
```

Potrzebujemy jeszcze nowego widoku `app/views/comments/create.js.erb`:

```
$("#comments").append("<%=j render @comment %>");
```

6. Logowanie przy użyciu Facebooka

Dodanie funkcjonalności uwierzytelniania użytkowników za pomocą Facebooka rozpoczynamy od edycji pliku `Gemfile`:

```
gem 'omniauth-facebook'
```

Przenosimy się do konsoli:

```
$ bundle
$ rails g migration add_facebook_uid_to_users facebook_uid:string
$ rake db:migrate
```

Od użytkowników logujących się przez facebooka nie powinniśmy wymagać hasła. Nadpisujemy metodę w modelu `app/models/user.rb`:

```
7  def password_required?
8    new_record? && facebook_uid.nil?
9  end
```

W tym samym modelu dodajemy obsługę mechanizmu OmniAuth:

```
2  devise :database_authenticatable, :validatable, :omniauthable
```

Teraz dodajemy niezbędne dane do konfiguracji OAuth w pliku `config/initializers/devise.rb`:

```
217 config.omniauth :facebook, '514730808568709',
218                 '8167fc00e25c0a0ffd1cc3e2e252f051', scope: 'email'
```

Kolej na dostosowanie routingu (`config/routes.rb`):

```
2  devise_for :users, controllers: {omniauth_callbacks: 'omniauth_callbacks'}
```

Musimy wygenerować kontroler z callbackiem z facebooka. W konsoli:

```
$ rails g controller omniauth_callbacks --assets=false
```

Poniżej prawidłowa implementacja stworzonego kontrolera `app/controllers/omniauth_callbacks_controller.rb`

```ruby
class OmniauthCallbacksController < Devise::OmniauthCallbacksController
  def facebook
    auth = request.env['omniauth.auth']

    @user = User.find_or_create_by_facebook_uid(auth.uid,
                email: auth.info.email, name: auth.extra.raw_info.name)

    if @user.persisted?
      set_flash_message(:notice, :success, kind: 'Facebook')
      sign_in_and_redirect @user
    else
      redirect_to root_path, alert: 'Something went wrong'
    end
  end
end
```

Dodajemy jeszcze link do strony logowania: `app/views/layouts/application.html.erb`

```erb
<%= link_to 'Sign in via FB', user_omniauth_authorize_path(:facebook) %>
```

7. Paperclip - upload obrazków

Standardowo rozpoczynamy od dodania nowej zależności w pliku `Gemfile`:

```
gem 'paperclip'
```

Następnie w konsoli wykonujemy kolejno polecenia:

```
$ bundle
$ rails g paperclip post image
$ rake db:migrate
```

Teraz musimy w modelu umiescić informację o powiązaniu z obrazkiem (`app/models/post.rb`):

```
3  attr_accessible :title, :content, :image
4  has_attached_file :image, styles: { medium: '400x400>' }
```

Dodajemy odpowiednie pole w formularzu wysyłania posta `app/views/admin/posts/_form.html.erb`:

```
11 <% if @post.image.exists? %>
12   <p><%= image_tag @post.image %></p>
13 <% end %>
14 <p>
15   <%= f.label :image %><br />
16   <%= f.file_field :image %>
17 </p>
```

Teraz możemy wyświetlić obrazek na dwóch widokach:
`app/views/posts/index.html.erb`:

```
6  <p><%= image_tag post.image.url(:medium) if post.image.exists? %></p>
```

`app/views/posts/show.html.erb`:

```
11 <p><%= image_tag @post.image.url(:medium) if @post.image.exists? %></p>
```