Конспект RubySchool.us [6]

Урок 36

HTTP запросы:

- GET /cart?...&...
- POST
- PUT
- DELETE

GET, POST можно отправлять с помощью form

PUT, DELETE через POST с добавлением скрытой переменной.

Паттерн REST - передача состояния объекта

В REST существует 7 различных методов с помощью которых можно управлять объектами:

- index
- show
- new
- create
- edit
- update
- destroy

REST product:

- index - /products - все наши продукты - get
- show - /products/1 - конкретный продукт - get
- new - /products/new - вывод формы для создания - get
- create - /products - создание продукта, обращ. к params и в БД создаёт запись - post
- edit - /products/1/edit - возвращает форму для редактирования определённого продукта - get
- update - /products/1 - put
- destroy - /produts/1 - delete

> Task Them All - Painless to-do lists to organize your [remote] workers

Продолжаем PizzaShop:

Очистим корзину после того как заказ был отправлен и добавим кнопку очистки корзины.

```
# + post '/cart'

# если корзина пустая
if @items.length == 0
  return erb "В корзине нет товаров"
end
```

Минусы созданного PizzaShop:

- модели из app.rb надо вынести в отдельный каталог
- много несвязанных между собой get, post в одном файле
- вспомогательный метод (хелпер) в этом же app.rb
- представления не в подкаталогах (как в рейлс)
- бардак с url
- в js дублирование кода
- нет тестов (в рейлс для всего существуют тесты)

Далее будем писать на Rails и постепенно эти минусы будут уходить.

Урок 37

Задача: посчитать все слова в файле:

Для этого используем хеш

```
f = File.open 'file.txt', 'r'

@hh = {}

def add_to_hash word
  if !word.empty?
    word.downcase!

    cnt = @hh[word].to_i
    cnt += 1

    @hh[word] = cnt
  end
end

f.each_line do |line|
  arr = line.split(/\s|\n|\.|,|:|;/)
  arr.each { |word| add_to_hash(word) }
end

f.close

@hh.each do |k, v|
  puts "#{v} - #{k}"
end
```

Продолжаем Rails

```
rails new blog
```

```
rails s
```

Запуск окружения (development, test, production)

```
rails s -e development
```

rails generate

rails generate [GEN] [параметры]

```
rails g
```

Генераторы: - controller - model - ...

```
rails g controller home index
```

home - контроллер index - экшен

Экшен - это обработчик какого-либо URL

Создадим модель:

```
rails g model Article title:string text:text
```

```
rake db:migrate
```

База данных находится в каталоге /db по адресу /db/development.sqlite

rake routes

Эта команда из каталога конфигурации берёт файл /config/routes.rb и нам его выводит.

```
rake routes
```

> https://guides.rubyonrails.org/routing.html

Внесём изменения в файл /config/routes.rb

```
Rails.application.routes.draw do
  get 'home/index'

  resources :articles
end
```

и снова посмотрим через:

```
rake routes
```

Выведет:

```
Prefix       Verb    URI Pattern                  Controller#Action

home_index   GET     /home/index(.:format)        home#index
articles     GET     /articles(.:format)          articles#index
POST                 /articles(.:format)          articles#create
new_article  GET     /articles/new(.:format)      articles#new
edit_article GET     /articles/:id/edit(.:format) articles#edit
article      GET     /articles/:id(.:format)      articles#show
             PATCH   /articles/:id(.:format)      articles#update
             PUT     /articles/:id(.:format)      articles#update
             DELETE  /articles/:id(.:format)      articles#destroy
```

Открыв http://localhost:3000/articles/new выпадет ошибка Routing Error. uninitialized constant ArticlesController

Т.к. у нас нет пока контроллера articles который отвечает за эту сущность.

Создадим контроллер:

```
rails g controller articles
```

> В /app/controllers/ наши контроллеры

Открыв http://localhost:3000/articles/new выпадет ошибка Unknown action The action 'new' could not be found for ArticlesController

Добавим экшен new в наш контроллер /app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController

  def new
  end

end
```

Открыв http://localhost:3000/articles/new выпадет ошибка ActionController::UnknownFormat in ArticlesController#new. ArticlesController#new is missing a template for this request format and variant.

Т.к. отсутствует шаблон (представление)

Представления расположены в каталоге/app/views/

Создадим файл /app/views/new.html.erb

```
touch /app/views/new.html.erb
```

> Всё вышеописанное делается с помощью одной команды, сейчас мы просто разбираемся в работе Rails.

Далее, в /app/views/new.html.erb добавим поля:

> Синтаксис похож на sinatra, но есть и отличия

```
<h1>New article</h1>

<%= form_for :article do |f| %>
test
<% end %>
```

<%= form_for %> - форм билдер. Существуют разные форм билдеры

Создаст форму (см. код страницы):

```
<form action="/articles/new" accept-charset="UTF-8" method="post"><input name="utf8" type="hidden" value="&#x2713;" /><input type="h
test
</form>
```

Выведем наши поля формы:

```
<h1>New article</h1>
<%= form_for :article, url: '/articles' do |f| %>
  <p><%= f.label :title %>
  <%= f.text_field :title %></p>

  <p><%= f.label :text %>
  <%= f.text_area :text %></p>

  <p><%= f.submit %></p>
<% end %>
```

Создали форму, но при нажати на кнопку submit ничего не произойдёт, т.к. у new у нас метод GET. (было <%= form_for :article do |f| %>)

Для POST используется экшен create. Нам надо перенаправить форму на этот URL /articles

Добавим к уже существующему коду:

```
<%= form_for :article, url: '/articles' do |f| %>
```

или так:

```
<%= form_for :article, url: articles_path do |f| %>
```

Добавим в app/controllers/articles_controller.rb экшен:

```
  def create
  end
```

Теперь при нажатии на кнопку будет выпадать ошибка: No template found for ArticlesController#create, т.к. отсутствует представление create

Попробуем вывести на экран параметры, которые нам передаются.

```
  def create
    render plain: params[:article].inspect
  end
```

У нас есть create и к нему идут обращения.

render - функция plain: params[:article].inspect - параметр функции plain: - ключ хеша params[:article].inspect - значение хеша

Домашнее задание:

- сделать страницу /home/contacts с формой для контактов (обычная как в синатре)
- сделать так, чтобы на сервер передавались email и message из формы контактов

Подсказка:

```
rails g controller ...
```

Сделать простым способом без REST.

Урок 38

Разбор вопросов на интервью:

Задача: перевернуть все слова в предложении:

```
string = "В лесу родилась ёлочка в лесу она росла"

puts string.split(/ /).reverse.join(' ')
```

На собеседовани при постановке тестовой задачи всегда уточнять требования.

Продолжаем разбор Rails. Создадим /contacts в нашем блоге:

http://localhost:3000/contacts

Routing Error. No route matches [GET] "/contacts"

```
rails g controller contacts
```

Нам нужно: - получение страницы с сервера (форма с текстовыми полями) - get (new) - отправка данных - post (create)

Мы хотим добавить только new и create, поэтому добавим:

```
resource :contacts, only: [:new, :create]
```

Добавим в /config/routes.rb

```
Rails.application.routes.draw do
  get 'home/index'

  resource :contacts, only: [:new, :create]
  resources :articles
end
```

Создадим представление: /app/views/contacts/new.html.erb

Откроем http://localhost:3000/contacts/new

Далее создадим модель, определимся с формой, полями:

```
rails g model Contact email:string message:text
```

ActiveRecord создал миграцию и файл модели + юнит-тест.

- /db/migrate/20181210205054_create_contacts.rb
- /app/models/contact.rb

```
rake db:migrate
```

У Ruby on Rails есть консоль. Запустим её:

```
rails console
```

и введём в неё:

```
Contact.all
```

Чтобы узнать какие свойства (поля) у сущности (модели) введём:

```
Contact.attribute_names
```

Продолжим делать блог:

Добавим в /app/views/contacts/new.html.erb:

```
<h2>Contact us</h2>

<%= form_for :contact, url: contacts_path do |f| %>
<p>
  <%= f.label :email %>
  <%= f.text_field :email %>
</p>
<p>
  <%= f.label :message %>
  <%= f.text_area :message %>
</p>

<p>
  <%= f.submit 'Send message' %>
</p>

<% end %>
```

Откроем http://localhost:3000/contacts/new

- :new получает GET
- :create получает POST

> Изучи ссылку: Rails Routing from the Outside In — Ruby on Rails Guides https://guides.rubyonrails.org/routing.html
>
> Изучи ссылку: Rusrails: Роутинг в Rails http://rusrails.ru/rails-routing
>
> Rails Routing from the Outside In — Ruby on Rails Guides https://guides.rubyonrails.org/routing.html#singular-resources

Поменять название кнопки отправки формы можно так:

```
<%= f.submit 'Send message' %>
```

> Для contact используем в единственном числе. Если у нас эта сущность будет одна и не надо выводить /contacts/1 .. /contacts/2 ..., то используем resource. Если много сущностей, используем resources.

Добавим предствление /app/views/contacts/create.html.erb

Запишем отправленные данные через форму в базу данных.

Откроем /app/controllers/contacts_controller.rb и запишем код, но...

```
class ContactsController < ApplicationController

  def new
  end

  def create
    @contact = Contact.new(params[:contact])
    @contact.save
  end

end
```

При обращении к этим параметрам будет выпадать ошибка, т.к. аттрибуты запрещены (связано с безопасностью). Нам надо их разрешить, для этого используется специальный синтаксис:

```
class ContactsController < ApplicationController

  def new
  end

  def create
    @contact = Contact.new(contact_params)
    @contact.save
  end

  private

  def contact_params
    params.require(:contact).permit(:email, :message)
  end

end
```

Если мы добавим ещё параметр, то и его надо будет указать в params.require(:contact).permit(:email, :message)

Теперь мы можем добавить запись в БД через форму http://localhost:3000/contacts/new

Проверим в rails console:

```
Contact.all
```

Сделаем валидацию нашей модели. Модели находятся в каталоге /app/models/

Валидацию надо добавить в /app/models/contact.rb

```
class Contact < ApplicationRecord
  validates :email, presence: true
  validates :message, presence: true
end
```

и исправим код в контроллере /app/controllers/contacts_controller.rb:

```
def create
  @contact = Contact.new(contact_params)
  if @contact.valid?
    @contact.save
  else
    render action: 'new'
  end
end
```

Если при заполнении формы будет ошибка валидаци, то форм билдер нам подскажет, обернув неправильно заполненное поле в div:

```
<div class="field_with_errors"></div>
```

Оформление можно оформить через CSS.

Добавим стиль в файл /app/assets/stylesheets/application.css

```
.field_with_errors input,
.field_with_errors textarea {
  border: 3px solid red;
}
```

Есть и другие форм-билдеры, мы использовали стандартный form_for, есть ещё Simple form. Для этого подключается отдельный гем.

Сейчас у нашего блога при заходе на http://localhost:3000/contacts выпадает Routing Error. No route matches [GET] "/contacts"

Сделаем так, чтобы по этому адресу у нас открывалась форма из contacts/new

Есть 2 способа.

1. Откроем /config/routes.rb и добавим код:

```
  get 'contacts' => 'contacts#new'
```

Обратить внимание, что при этом мы удалили :new из строки:

```
  resource :contacts, only: [:create]
```

Весь файл:

```
Rails.application.routes.draw do
  get 'home/index'

  get 'contacts' => 'contacts#new'
  resource :contacts, only: [:create]
  resources :articles
end
```

2. Альтернативный способ при котором 1 строка отвечает за 1 сущность:

```
Rails.application.routes.draw do
  get 'home/index'

  resource :contacts, only: [:new, :create], path_names: { :new => '' }
  resources :articles
end
```

Перенаправление по базовому пути: path_names: { :new => '' }

Сделаем вывод списка ошибок:

Откроем /app/views/contacts/new.html.erb и добавим код:

```
<p><%= @contact.errors.full_messages %></p>
```

В app/controllers/contacts_controller.rb можно определить переменную:

```
def new
  @contact = Contact.new
end
```

Либо записать вот так (в /app/views/contacts/new.html.erb):

```
<% if @contact && @contact.errors.any? %>
  <p><%= @contact.errors.full_messages %></p>
<% end %>
```

Тогда этот блок при отсутствии переменной (т.е. при /new) не будет отображаться на странице.

Оформим вывод ошибок в виде списка:

```
<% if @contact && @contact.errors.any? %>
  <ul>
    <% @contact.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
  </ul>
<% end %>
```

Сравним 2 таблицы

> https://guides.rubyonrails.org/routing.html#singular-resources и https://guides.rubyonrails.org/routing.html#crud-verbs-and-actions

(resource) singular-resources:

```
 HTTP Verb     Path              Controller#Action   Used for
GET           /geocoder/new     geocoders#new       return an HTML form for creating the geocoder
POST          /geocoder         geocoders#create    create the new geocoder
GET           /geocoder         geocoders#show      display the one and only geocoder resource
GET           /geocoder/edit    geocoders#edit      return an HTML form for editing the geocoder
PATCH/PUT     /geocoder         geocoders#update    update the one and only geocoder resource
DELETE        /geocoder         geocoders#destroy   delete the geocoder resource
```

(resources) crud-verbs-and-actions:

```
 HTTP Verb     Path            Controller#Action   Used for
GET           /photos         photos#index        display a list of all photos
GET           /photos/new     photos#new          return an HTML form for creating a new photo
POST          /photos         photos#create       create a new photo
GET           /photos/:id     photos#show         display a specific photo
GET           /photos/:id/edit  photos#edit       return an HTML form for editing a photo
PATCH/PUT     /photos/:id     photos#update       update a specific photo
DELETE        /photos/:id     photos#destroy      delete a specific photo
```

- В resources есть метод выводящий всех ресурсов.
- Следующее отличие - это /show

Профиль у пользователя 1 - resource

Домашнее задание:

1. FizzBuzz тест: вывести список чисел от 1 до 100, если число делится на 3 писать Fizz, если число делится на 5 писать Buzz, если и на 3 и на 5 писать FizzBuzz иначе выводить просто само число.
2. Создать в нашем блоге страницы:

- /terms - условия использования - /about - о нашем блоге

3. Реализовать в блоге articles#create
4. Сделать редактирование статьи.
5. Сделать вывод списка всех статей.

Урок 39

resource и resources отвечают за REST маршруты приложения, они записываются в /config/routes.rb

resource (profile):

- показать - show
- создать - new (отобразить форму. GET), create (отправить форму. POST)
- редактировать - edit, update
- удалить - destroy

resources (articles):

- показать список - index
- показать - show
- создать - new (отобразить форму. GET), create (отправить форму. POST)
- редактировать - edit, update
- удалить - destroy

Продолжим делать блог

У нас есть http://localhost:3000/articles/new и http://localhost:3000/contacts

Нам нужно создать новую статью. Сделаем это. Откроем /app/controllers/articles_controller.rb и доделаем метод create

Посмотрим какие поля существуют в нашей сущности Article

```
 rails console
Article.attribute_names
```

=> ["id", "title", "text", "created_at", "updated_at"]

Внесём изменения в /app/controllers/articles_controller.rb:

```
class ArticlesController < ApplicationController

  def new
  end

  def create
    @article = Article.new(article_params)
    if @article.valid?
      @article.save
      redirect_to @article
    else
      render action: 'new'
    end
  end

  private

  def article_params
    params.require(:article).permit(:title, :text)
  end

end
```

Теперь нам надо создать представление для create. Создадим: /app/views/articles/create.html.erb

```
<h2>Спасибо!</h2>
<p>Статья создана</p>
```

Посмотреть список статей в консоли rails:

```
rails console
Article.all
```

Добавим ссылку на все статьи в /app/views/articles/create.html.erb

Так не делают:

```
<a href="/articles">Показать все статьи</a>
```

Делают так:

```
<%= link_to "Показать все статьи", articles_path %>
```

> Для борьбы с двойным сабмиттом существует паттерн PRG (Post Redirect Get)

Для этого мы добавили строку:

```
redirect_to @article
```

У нас происходит редирект на show поэтому представление create нам теперь не нужно, его можно удалить.

Добавим в app/controllers/articles_controller.rb:

```
  def show
    @article = Article.find(params[:id])
  end
```

Создадим представление /app/views/articles/show.html.erb

```
<h1><%= @article.title %></h1>

<p><%= @article.text %></p>
```

Вывод списка статей

Список статей будет доступен по адресу http://localhost:3000/articles

Добавим в контроллер /app/controllers/articles_controller.rb экшен index:

```
def index
  @articles = Article.all
end
```

И, создадим вьюху /app/views/articles/index.html.erb

```
<% @articles.each do |article| %>
  <h3><%= article.title %></h3>
  <p><%= article.text %></p>
  <p><%= link_to  "Show article", article_path(article) %> | <%= link_to  "Edit article", edit_article_path(article) %></p>
  <hr>
<% end %>
```

Редактирование статьи (edit, update):

Кнопка на редактирование:

```
<%= link_to "Edit article", edit_article_path(article) %>
```

Добавим в /app/controllers/articles_controller.rb:

```
def edit
  @article = Article.find(params[:id])
end
```

И, создадим вьюху /app/views/articles/edit.html.erb:

```
<h1>Edit article</h1>

<%= form_for :article, url: article_path(@article), method: :patch do |f| %>
<p>
  <%= f.label :title %>
  <%= f.text_field :title %>
</p>
<p>
  <%= f.label :text %>
  <%= f.text_area :text %>
</p>

<p>
  <%= f.submit %>
</p>

<% end %>
```

Добавим в контроллер /app/controllers/articles_controller.rb экшен update:

```
def update
  @article = Article.find(params[:id])

  if @article.update(article_params)
    redirect_to @article
  else
    render action: 'edit'
  end
end
```

Контроллер и роутинг статических страниц

```
rails g controller Pages
```

Создаётся контроллер /app/controllers/pages_controller.rb, внесём код:

```
class PagesController < ApplicationController

  def terms
  end

  def about
  end

end
```

И, пропишем маршруты в /config/routes.rb:

```
  get 'terms' => 'pages#terms'
 get 'about' => 'pages#about'
```

И, создадим вьюхи /app/views/pages/terms.html.erb и /app/views/pages/about.html.erb

> Изучи ссылку: Rusrails: Командная строка Rails http://rusrails.ru/a-guide-to-the-rails-command-line

Домашнее задание:

- переписать из rake routes
- сделать destroy (delete)

> Ссылка на репозиторий с учебным блогом на Rails: https://github.com/krdprog/RailsBlog-rubyschool

Урок 40

Повторение:

Чтобы показать обычные статические страницы (типа /terms, /about) нам надо:

- создать контроллер (rails g controller ...)
- добавить actions (методы terms, about)
- добавить views (terms.html.erb, about.html.erb)
- добавить в routes.rb маршруты (get 'terms' => 'pages#terms')

В синатре это занимало 2 действия:

- маршрут
- views

Создание сущности:

- создание модели (rails g model ...)
- rake db:migrate
- добавить маршрут в routes.rb - resource или resources
- добавить контроллер
- в контроллер добавить actions (index, show, new, edit, create, update, destroy)
- добавить views

Продолжим делать RailsBlog, доделаем удаление Article:

Чтобы удалить сущность, надо сделать 2 действия:

- найти сущность по id
- удалить

Добавим в articles_controller.rb метод destroy:

```
def destroy
  @article = Article.find(params[:id])
  @article.destroy

  redirect_to articles_path
end
```

Добавим в файл /views/articles/index.html.erb кнопку удаления статьи:

```
<%= link_to "Destroy", article_path(article), method: :delete %>
```

или с подтверждением:

```
<%= link_to "Destroy", article_path(article), method: :delete, data: { confirm: 'Действительно удалить?'} %>
```

К тегу ссылки на удаление добавится атрибут data-method="delete"

Файл turbolinks обрабатывает атрибуты data-* встречающиеся в коде html

Разбор data-method="delete"

- скрипт сканирует страницу
- ищет атрибут data-method="delete"
- создаёт форму, которая будет отправлять с помощью метода delete на URL /articles/2 через POST
- устанавливает свой обработчик, при котором при нажатии на ссылку вызывается сабмит формы

Полезная особенность генераторов, возможность указывать reference columns - делать ссылки на другие сущности

```
rails g model photo album:references
```

> Изучить ссылку: https://railsguides.net/advanced-rails-model-generators/

Добавление комментариев к статьям:

one-to-many:

```
Article -1------*- Comment
```

Создадим модель Comment:

```
rails g model Comment author:string body:text article:references
rake db:migrate
```

Это добавит в файл /models/comment.rb:

```
class Comment < ApplicationRecord
  belongs_to :article
end
```

А, вот содержимое миграции (/db/migrate/12312314_create_comments.rb):

```
class CreateComments < ActiveRecord::Migration[5.2]
  def change
    create_table :comments do |t|
      t.string :author
      t.text :body
      t.references :article, foreign_key: true

      t.timestamps
    end
  end
end
```

Посмотрим в базе данных:

```
cd db
sqlite3 development.sqlite3
```

```
select * from Articles;
.tables
select * from Comments;
```

Посмотреть через sqlite3, какие поля есть у сущности:

```
pragma table_info(articles);
```

Идём дальше, чтобы добавить к статьям комментарии нам надо в /models/article.db добавить:

```
class Article < ApplicationRecord
  has_many :comments
end
```

Таким образом мы связали 2 сущности между собой.

у нас в /config/routes.rb есть строка:

```
resources :articles
```

Допишем и сделаем **вложенный маршрут**:

```
resources :articles do
  resources :comments
end
```

Команда rake routes покажет нам обновлённую карту маршрутов.

Теперь добавим контроллер:

```
rails g controller Comments
```

Далее, создадим методы в контроллере.

Для комментариев нам нужен один метод - create

Посмотрим в rails console:

```
Comment.all
@article = Article.find(1)
@article.comments.create({ author: 'Foo', body: 'Bar' })
@article.comments
```

Создадим метод create в /app/controllers/comments_controller.rb:

```
class CommentsController < ApplicationController
  def create
    @article = Article.find(params[:article_id])
    @article.comments.create(comment_params)

    redirect_to article_path(@article)
  end

  private

  def comment_params
    params.require(:comment).permit(:author, :body)
  end

end
```

Добавление формы в представление статьи:

> Изучить ссылку: https://guides.rubyonrails.org/association_basics.html
>
> Изучить ссылку: https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html

```
<p>
<%= form_for([@article, @comment]) do |f| %>
<% end %>
</p>
```

build:

```
<p>
<%= form_for([@article, @article.comments.build]) do |f| %>
<% end %>
</p>
```

Добавим форму в представление (/app/views/articles/show.html.erb):

```
<p>
<%= form_for([@article, @article.comments.build]) do |f| %>

  <p>
    <%= f.label :author %>
    <%= f.text_field :author %>
  </p>
  <p>
    <%= f.label :body %>
    <%= f.text_area :body %>
  </p>

  <p><%= f.submit %></p>

<% end %>
</p>
```

Проверим форму, добавив данные. И, посмотрим в рейлс-консоли

```
Comment.last
Comment.all
@article = Article.find(1)
@article.comments
```

Домашнее задание:

- Сделать вывод комментариев на странице статьи
- Избавиться от ненужных маршрутов
- Сравнить код BarberShop на Rails и на Sinatra

Продолжение конспекта: Урок 41-45 - https://github.com/krdprog/rubyschool-notes/blob/master/rubyschool-notes-07.md

Содержание конспекта: