Конспект RubySchool.us [7]

Урок 41

Когда используется единственное число, а когда множественное?

- таблицы в БД - множественное число
- модели - единственное число
- контроллеры - множественное число

> В книге Rails. Гибкая разработка веб-приложений (Руби, Томас, Хэнссон) прочитать про соглашения об именах.

Типы связей

> http://rusrails.ru/active-record-associations

Тип 1 - * (one-to-many)

```
Article            |  Comment

has_many :comments |  belongs_to :article
id                 |  id, article_id
```

Тип 1 - 1 (one-to-one) - помогает нормализовать БД

```
Order              |  Address

has_one :address   |  belongs_to :order
id                 |  id, order_id
```

Нормализация, денормализация. Плюсы и минусы подходов. В варианте выше потребуется ещё 1 запрос к базе данных.

> Понятие complex_type - сложный тип (изучить)

Тип * - * (many-to-many)

```
Tag                              |  Article

таблица tags                     |  таблица articles
id                               |  id
has_and_belongs_to_many :articles | has_and_belongs_to_many :tags
```

для связи между ними создаётся ещё одна таблица tags_articles (tag_id, article_id)

> Изучить: http://www.rusrails.ru/active-record-associations#foreign_key

Вывод комментариев в представлении статьи:

Добавим в /app/views/articles/show.html.erb:

```
<h3>Комментарии:</h3>

<%= 'Без комментариев' if @article.comments.empty? %>

<% @article.comments.each do |comment| %>

<p><strong>Автор:</strong> <%= comment.author %></p>
<p><%= comment.body %></p>
<hr>

<% end %>
```

Ещё раз про ActiveRecord

Вспомним CRUD:

- Create - (new) - .create - .save
- Read - .where - .find(3), .all
- Update - (update)
- Delete - (destroy)

> https://github.com/rails/strong_parameters
>
> https://guides.rubyonrails.org/action_controller_overview.html#more-examples

Вспомним REST:

- resource
- resources

Можно вкладывать, получаются длинные URL:

```
resources :article do
  resources :comments
end
```

```
x = 2 != 3
puts x #=> true
```

Rspec - фреймворк для тестирования приложений

```
gem install rspec
```

Запуск тестов:

```
rspec
```

В Rspec существует 2 слова:

- describe
- it

Создадим и протестируем героя компьютерной игры.

> https://github.com/krdprog/rspec-demo-rubyschool - репозиторий Rspec-demo

Создадим файл hero.rb

```
class Hero

  def initialize(name, health=100)
    @name = name.capitalize
    @health = health
  end

  attr_reader :name

  def power_up
    @health += 10
  end

  def power_down
    @health -= 10
  end

  def hero_info
    "#{@name} has #{@health} health"
  end

end
```

Создадим файл hero_spec.rb:

```
require './hero'

describe Hero do

  it "has a capitalized name" do
    hero = Hero.new 'foo'

    expect(hero.name).to eq 'Foo'
  end

  it "can power up" do
    hero = Hero.new 'foo'

    expect(hero.power_up).to eq 110
  end

end
```

И, запустим тест:

```
rspec hero_spec.rb
```

Написание тестов в большом приложении - это вклад в будущее, защита приложения от ошибок.

Урок 42

Продолжим рассматривать Rspec и тестирование.

Добавим к нашим тестам в файле hero_spec.rb

```
  it "can power down" do
    hero = Hero.new 'foo'

    expect(hero.power_down).to eq 90
  end

  it "displays full hero info" do
    hero = Hero.new 'foo'

    expect(hero.hero_info).to eq "Foo has 100 health"
  end
```

У нас часто повторяется код hero = Hero.new 'foo', и это не совпадает с DRY (Don`t Repeat Yourself), ниже мы его оптимизируем, добавив before-do-end

Рефакторинг hero_spec.rb:

```
require './hero'

describe Hero do

  before do
    @hero = Hero.new 'foo'
  end

  it "has a capitalized name" do
    expect(@hero.name).to eq 'Foo'
  end

  it "can power up" do
    expect(@hero.power_up).to eq 110
  end

  it "can power down" do
    expect(@hero.power_down).to eq 90
  end

  it "displays full hero info" do
    expect(@hero.hero_info).to eq "Foo has 100 health"
  end

end
```

Тесты должны быть:

- надёжные (reliable) - дают тот же результат - без зависимостей от соединения, от БД и т.п.
- easy to write - если тест пишется не легко, то что-то не так с тем, что тестируем.
- easy to understand - лёгкие для понимания другими программистами.
- скорость не особо важна
- не важно DRY

Rspec Matchers

> Изучить ссылку: https://relishapp.com/rspec/rspec-expectations/docs/built-in-matchers

Сделаем ещё одно приложение и тест:

car.rb:

```
class Car

  Miles_Per_Gallon = 20

  attr_reader :fuel

  def initialize
    @fuel = 0
  end

  def add_fuel amount
    @fuel += amount
  end

  # Как далеко мы сможем проехать:
  def range
    @fuel * Miles_Per_Gallon
  end

end

# car = Car.new
# car.add_fuel 10
# puts "Range is #{car.range}"
```

car_spec.rb:

```
require "./car"

describe Car do
  it "must return range" do
    # arrange
    car = Car.new
    # act
    car.add_fuel 10
    # assert
    expect(car.range).to eq 200
  end
end
```

Структура тестов:

```
# arrange
# act
# assert
```

```
# arrange - подготовка объекта для проведения теста
  car = Car.new
# act - действие
  result = car.add_fuel 10
# assert - проверка действия
  expect(result) ...
```

> Заметка по Issue на Github. У issue есть номер (например #12), если при коммите указать этот номер, то issue сама закроектся и пометится как выполненная.

Devise

Devise - гем для авторизации

Добавим в Gemfile:

```
gem 'devise'
```

```
bundle
```

Вывести в терминал список генераторов в системе:

```
rails g
```

```
Devise:
  devise
  devise:controllers
  devise:install
  devise:views
```

Введём:

```
rails g devise:install
```

План, как подключать devise:

1. gem 'devise' в Gemfile
2. rails g devise:install

Проверить есть ли строка в файле config/environments/development.rb:

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

Проверить, что в /config/routes.rb указано:

```
root to: "home#index"
```

Добавить в app/views/layouts/application.html.erb для флеш-сообщений:

```
<p class="notice"><%= notice %></p>
<p class="alert"><%= alert %></p>
```

Для кастомизации форм:

```
rails g devise:views
```

Далее, создадим модель пользователя:

```
rails g devise User
```

Devise создал параметры: e-mail, зашифрованный пароль, токен для сброса пароля.

Наберём:

```
rake db:migrate
```

И, запустим:

```
rails s
```

Задача: чтобы мы могли просматривать статьи, но не могли их создавать.

- http://localhost:3000/articles
- http://localhost:3000/articles/new

Откроем /app/controllers/articles_controller.rb и добавим:

> Примечание: начиная с Rails 5 синтаксис before_filter устарел и заменён на before_action

```
before_action :authenticate_user!
```

Добавим в /app/views/layouts/application.html.erb:

```
<p><a href="/users/sign_in">Sign In</a> | <a href="/users/sign_out" data-method="delete">Sign Out</a></p>
```

Далее, мы заменим эти ссылки на ссылки с именноваными маршрутами.

> Документация по гему devise - https://github.com/plataformatec/devise
>
> Статья на Хабре по devise - https://habr.com/ru/post/208056/
>
> Посмотреть примеры - https://github.com/plataformatec/devise/wiki/Example-applications

Урок 43

Заменим жёстко прописанные ссылки, на ссылки с именованными маршрутами:

```
<p><%= link_to "Sign In", new_user_session_path %> | <%= link_to "Sign Out", destroy_user_session_path, method: :delete %></p>
```

Далее, выведем имя пользователя при авторизации, и уберём Sign Out ссылку, когда мы не авторизованы.

> https://github.com/plataformatec/devise#controller-filters-and-helpers

```
<% if user_signed_in? %>
  Hello, <%= current_user.email %> | <%= link_to "Sign Out", destroy_user_session_path, method: :delete %>
<% else %>
  <%= link_to "Sign In", new_user_session_path %>
<% end %>
```

Авторизация, сессии

- Аутентификация - проверка пользователя и пароля
- Авторизация - наделение определёнными правами в зависимости от роли (юзер/админ)

```
       Login, password
User ------------------> Server
     <------------------
       Cookies (*)
```

/config/initializers/devise.rb

```
# config.secret_key =
```

Механизм сессий

```
session['key'] = 'value'
```

Cookie выдаётся при первом обращении к серверу, и затем без авторизации идёт обмен данными.

JSON - универсальный формат данных

> https://ru.wikipedia.org/wiki/JSON

Следующий пример показывает JSON-представление объекта, описывающего человека. В объекте есть строковые поля имени и фамилии, объект, описывающий адрес, и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```

Добавление username в наш блог, и вставка вместо поля автора комментария, username залогиненого пользователя:

```
rails g migration add_username
```

Добавим в существующую таблицу новый столбец.

> https://api.rubyonrails.org/classes/ActiveRecord/Migration.html

/db/migrate/20190129063426_add_username.rb

```
class AddUsername < ActiveRecord::Migration[5.2]
  def change
    add_column :users, :username, :string
    add_index :users, :username, unique: true
  end
end
```

```
rake db:migrate
```

Индекс add_index:

Индекс - увеличивается время вставки, но уменьшается время выборки по определённому полю.

Devise надо указать какие дополнительные параметры можно задать.

Все контроллеры наследуются от базового контроллера ApplicationController и чтобы задать для всех контроллеров один параметр, надо прописать это в ApplicationController.

before_action (в ранних версиях Rails это было before_filter), фильтрует методы в контроллерах до того как их обработать.

/app/controllers/application_controller.rb:

```
class ApplicationController < ActionController::Base
  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up, keys: [:username])
  end
end
```

Далее, отредактируем форму регистрации Sign Up:

Сгенерируем набор views:

```
rails g devise:views
```

Добавим в /app/views/devise/registrations/new.html.erb код поля ввода username:

```
<div class="field">
  <%= f.label :username %><br />
  <%= f.text_field :username, autocomplete: "username" %>
</div>
```

И, выведем имя вместо e-mail в /app/views/layouts/application.html.erb:

```
<% if user_signed_in? %>
  Hello, <%= current_user.username %> | <%= link_to "Sign Out", destroy_user_session_path, method: :delete %>
<% else %>
  <%= link_to "Sign In", new_user_session_path %>
<% end %>
```

Сделаем так, чтобы при комментировании статьи автор указывался тот, кто залогинен

Авторизация только для создания и редактирования статьи /app/controllers/articles_controller.rb:

```
before_action :authenticate_user!, :only => [:new, :create, :edit, :update, :destroy]
```

И, для комментирования /app/controllers/comments_controller.rb:

```
before_action :authenticate_user!, :only => [:create]
```

Домашнее задание:

- Найти и изучить, что такое индексы в БД.
- Сделать так, чтобы комментарии оставлялись под именем залогиненого пользователя.

> https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-indeksy/
>
> http://www.sql.ru/articles/mssql/03013101indexes.shtml
>
> https://habr.com/ru/post/247373/
>
> https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

Урок 44

На upwork.com указывать, что все fees берёт на себя заказчик. Т.е. при $30 будет отображаться $33 (пример требующий проверки).

> Далее в этом уроке рассматривается Bootstrap. Конспекты по нему не фиксировались.

Домашнее задание:

- Читать про хелперы
- Внедрить в RailsBlog - bootstrap
- Убрать поле author из view комментария, и заменить на имя залогиненого (если не сделано на уроке 43)

Урок 45

Принцип работы метода params в контроллерах: в params хранятся все параметры которые передаются из браузера в приложение.

```
private

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

> https://stackoverflow.com/questions/18424671/what-is-params-requireperson-permitname-age-doing-in-rails-4
>
> https://api.rubyonrails.org/classes/ActionController/Parameters.html#method-i-require
>
> https://api.rubyonrails.org/classes/ActionController/Parameters.html#method-i-permit
>
> https://api.rubyonrails.org/classes/ActionController/Parameters.html

```
Browser ===> Server ===> Controller ===> ActiveRecord ===> Database
                                            ||
                                            ===> Error
```

Правило: никогда не верьте тому, что передаёт клиент.

Are redirect_to and render exchangeable?

> https://stackoverflow.com/questions/7493767/are-redirect-to-and-render-exchangeable

Если используется render когда юзер обновляет страницу, то он засабмитит предыдущий POST-запрос, это может отправить дополнительные данные.

Если используется redirect_to, то это будет выгдядеть как редирект.

Post/Redirect/Get (PRG) pattern:

> https://en.wikipedia.org/wiki/Post/Redirect/Get

Хелперы:

Все хелперы расположены в каталоге /app/helpers/

Для каждого контроллера существует хелпер, можно вызывать различные методы из этих хелперов, из разных представлений. Хелпер - работает между контроллером и представлением. Чтобы не вставлять код в представление.

Чтобы не дублировать код в каждом представлении.

Логику в представлениях писать неправильно, надо выносить в хелперы. Представления предназначены для того, чтобы отображать данные. Нехорошо размазывать логику по всему приложению.

Один хелпер создаётся для одного контроллера, но все хелперы доступны всем контроллерам.

Существуют также **встроенные хелперы** в Rails, например **debug**, выведет список параметров.

```
<%= debug(params) %>
```

Ещё посмотреть:

```
<%= simple_format(@foo) %>
```

И, см. автоматическая подсветка ссылок - **autolinks**

А, также **truncate - если есть длинная строка, то обрезается под указанный размер:**

```
<%= truncate(@foo, length: 20) %>
```

И, уже известный нам хелпер link_to.

> См. ещё: http://rusrails.ru/action-view-overview
>
> https://guides.rubyonrails.org/form_helpers.html

Как устроен процесс разработки в компании (CI, CD)

- CI - continuous integration - непрерывная интеграция - автотест при коммитах в общий репозиторий
- CD - continuous delivery - непрерывная доставка - автозаливка на веб-сервер

```
programmer  owner
   |   ______|
hipchat _______ github -------
   |___________ test server__|
                 |
                www (staging) --- QA
```

Integration tests - watir, selenium

> https://github.com/watir/watir
>
> https://github.com/SeleniumHQ/selenium
>
> KANBAN-доска: http://kanbanflow.com/
>
> http://trello.com/

Vargant

> https://www.vagrantup.com/
>
> https://github.com/rails/rails-dev-box

Домашнее задание:

- поставить Vargant
- привязать сущность article в блоге к пользователю. Сделать так, чтобы другие пользователи не могли редактировать статьи.

Продолжение конспекта: Урок 46-50 - https://github.com/krdprog/rubyschool-notes/blob/master/rubyschool-notes-08.md

Содержание конспекта: