Конспект RubySchool.us [8]

Урок 46

Тестирование моделей

Добавить в Gemfile нашего RailsBlog:

```
group :test, :development do
  gem 'rspec-rails'
  gem 'shoulda-matchers'
  gem 'capybara'
end
```

```
bundle install
```

Настройка Rspec для Rails

```
rails g rspec:install
```

Настроим Shoulda-matchers, добавив в spec/rails_helper.rb:

```
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    with.test_framework :rspec
    with.library :rails
  end
end
```

> Тестирование проводится в разрабатываемом учебном проекте RailsBlog - https://github.com/krdprog/RailsBlog-rubyschool

- Создадутся новые каталоги и хелперы. Создастся каталог /spec
- Создадим для тестирования моделей каталог /spec/models

Модель, которую будем тестировать (/app/models/contact.rb):

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

Создадим тест для модели: создать файл /spec/models/contact_spec.rb

```
require 'rails_helper'

describe Contact do
  it { should validate_presence_of :email }
  it { should validate_presence_of :message }
end
```

Запустим тест:

```
rake spec
```

should - должно

Shoulda matchers:

> https://relishapp.com/rspec/rspec-expectations/docs/built-in-matchers
>
> https://matchers.shoulda.io/
>
> http://matchers.shoulda.io/docs/v3.1.3/
>
> https://github.com/thoughtbot/shoulda-matchers

have_many:

> http://matchers.shoulda.io/docs/v3.1.3/Shoulda/Matchers/ActiveRecord.html#have_many-instance_method

Создадим тест /spec/models/article_spec.rb:

```
require 'rails_helper'

describe Article do
  it { should have_many :comments }
end
```

Создадим тест /spec/models/comment_spec.rb

```
require 'rails_helper'

describe Comment do
  it { should belong_to :article }
end
```

Вложенный describe - повышает читаемость тестов

/spec/models/article_spec.rb:

```
require 'rails_helper'

describe Article do
  describe "validations" do
    it { should validate_presence_of :title }
    it { should validate_presence_of :text }
  end

  describe "assotiations" do
    it { should have_many :comments }
  end
end
```

Принято писать для НЕ методов:

```
describe "something" do
```

А, для instance методов:

```
describe "#method" do
```

Для class методов (self.method):

```
describe ".method" do
```

Добавим в /app/models/article.rb

```
def subject
  title
end
```

И, протестируем этот метод, добавим тест в /spec/models/article_spec.rb

```
require 'rails_helper'

describe Article do
  describe "#subject" do
    it "returns the article title" do
      article = create(:article, title: 'Foo Bar')

      expect(article.subject).to  eq 'Foo Bar'
    end
  end
end
```

Чтобы этот тест заработал, нужен гем factory bot

gem Factory Girl (устарел) –> Factory Bot (использовать)

Помогает при тестировании, чтобы не создавать объекты для теста, создаётся фабрика, и она будет создавать нам объекты для теста.

ВАЖНОЕ ЗАМЕЧАНИЕ!!! Гем factory girl - устарел, надо использовать factory bot

> https://github.com/thoughtbot/factory_bot/blob/v4.9.0/UPGRADE_FROM_FACTORY_GIRL.md
>
> https://www.rubydoc.info/gems/factory_bot/file/GETTING_STARTED.md

```
DEPRECATION WARNING: The factory_girl gem is deprecated. Please upgrade to factory_bot. See https://github.com/thoughtbot/factory_bo
```

Добавить в Gemfile:

```
group :development, :test do
  gem "factory_bot_rails"
end
```

```
bundle install
```

И, добавить конфигурацию в /spec/rails_helper.rb:

```
RSpec.configure do |config|
  config.include FactoryBot::Syntax::Methods
end
```

Как создавать фабрики:

> https://www.rubydoc.info/gems/factory_bot/file/GETTING_STARTED.md#Defining_factories

Создадим каталог с фабриками - /spec/factories

И, создадим файл /spec/factories/articles.rb:

```
FactoryBot.define do
  factory :article do
    title { "Article title" }
    text { "Article text" }
  end
end
```

Преимущество использования factory bot для тестирования в том, что не нужна база данных.

> Как всё выглядит в RailsBlog см. https://github.com/krdprog/RailsBlog-rubyschool

Создадим в /app/models/article.rb метод last_comment и протестируем его:

```
def last_comment
  comments.last
end
```

Добавим в /spec/models/article_spec.rb

```
describe Article do
  describe "#last_comment" do
    it "returns the last comment" do

    end
  end
end
```

Создадим фабрику /spec/factories/comments.rb:

```
FactoryBot.define do
  factory :comment do
    author { "Chuck Norris" }
    sequence(:body) { |n| "Comment body #{n}" }
  end
end
```

Sequences (последовательности):

> https://www.rubydoc.info/gems/factory_bot/file/GETTING_STARTED.md#Sequences

См. документацию по create_list (через поиск по странице):

> https://www.rubydoc.info/gems/factory_bot/file/GETTING_STARTED.md

Добавим в /spec/factories/articles.rb:

```
FactoryBot.define do
  factory :article do
    title { "Article title" }
    text { "Article text" }

    # создаём фабрику для создания статьи с несколькими комментариями
    factory :article_with_comments do
      # после создания article
      after :create do |article, evaluator|
        # создаём список из 3-х комментариев
        create_list :comment, 3, article: article
      end
    end
  end
end
```

В /spec/models/article_spec.rb создадим статью с комментариями для тестирования:

```
require 'rails_helper'

describe Article do
  describe "#last_comment" do
    it "returns the last comment" do
      # создаём статью с 3 комментариями
      article = create(:article_with_comments)

      # проверка
      expect(article.last_comment.body).to eq "Comment body 3"
    end
  end
end
```

Домашнее задание:

- Добавить в сущность Article валидацию длины title в 140 символов и написать тесты.
- Добавить в сущность Article валидацию длины text в 4000 символов и написать тесты.
- В Comment добавить валидацию длины body в 4000 символов и написать тесты.

Урок 47

TODO: В конце урока требуется разобраться с парой ошибок. Оставил на потом, чтобы понять и решить как делать. Внимание.

Приёмочное тестирование (Acceptance Testing)

Проверка функциональности на соответствие требованиям. Отличие от юнит-тестов, что для этих тестов обычно существует план приёмочных работ. Юнит-тесты - проверка чтобы не сломалось.

> http://protesting.ru/testing/levels/acceptance.html

unit:

```
describe
  it
```

acceptance:

```
feature
  scenario
```

feature scenario - это фишка Capybara

Capybara

> https://www.rubydoc.info/github/teamcapybara/capybara/master

feature - особенность scenario - сценарий

Добавить в Gemfile:

```
group :test do
  gem 'capybara'
end
```

2 типа тестов:

- visitor_..._spec.rb - анонимный пользователь
- user_..._spec.rb - пользователь залогиненый в системе

Using Capybara with RSpec:

> https://github.com/teamcapybara/capybara#using-capybara-with-rspec

Пример: Для контактной формы существует 2 сценария:

1. Убедиться, что контактная форма существует.
2. Что мы можем эту форму заполнить и отправить

Проведём тестирование контактной формы в учебном приложении RailsBlog.

> https://github.com/krdprog/RailsBlog-rubyschool

Протестируем форму на http://localhost:3000/contacts

new_contacts_path - это именованный маршрут для /contacts

Создадим каталог /spec/features и создадим файл /spec/features/visitor_creates_contact_spec.rb:

```
require "rails_helper"

feature "Contact creation" do
  scenario "allows acess to contacts page" do
    visit new_contacts_path

    expect(page).to have_content 'Contact us'
  end
end
```

Работа с i18n (internationalization)

- Открыть файл /config/locales/en.yml

```
en:
  hello: "Hello world"
```

- Обязательно должны быть 2 пробела, иначе yml не заработает.

Настройка в Sublime Text (Preferences - Settings - Syntax Specific):

```
{
  "tab_size": 2,
  "translate_tabs_to_spaces": true
}
```

Можно создавать перевод для сайта и вызывать константы во views. Например, для русского языка можно создать /config/locales/ru.yml

```
# To use the locales, use `I18n.t`:
#
#     I18n.t 'hello'
#
# In views, this is aliased to just `t`:
#
#     <%= t('hello') %>
#
# To use a different locale, set it with `I18n.locale`:
#
#     I18n.locale = :es
```

Создадим в /config/locales/en.yml:

```
en:
  contacts:
    contact_us: "Contact Us!"
```

И, вызовем в представлении /app/views/contacts/new.html.erb:

```
<h2><%= t('contacts.contact_us') %></h2>
```

Применение i18n в Capybara: Исправим наш тест с учётом i18n файл /spec/features/visitor_creates_contact_spec.rb:

```
require "rails_helper"

feature "Contact creation" do
  scenario "allows acess to contacts page" do
    visit new_contacts_path

    expect(page).to have_content I18n.t('contacts.contact_us')
  end
end
```

Запустим тест:

```
rake spec
```

Следующий тест, создание самого сообщения:

Откроем страницу с формой заявки и откроем код формы, чтобы узнать id (будем использовать в тесте):

```
<input name="contact[email]" id="contact_email" type="text">
<textarea name="contact[message]" id="contact_message"></textarea>
```

Наш файл с тестом (features/visitor_creates_contact_spec.rb) будет выглядеть так:

```
require "rails_helper"

feature "Contact creation" do
  scenario "allows acess to contacts page" do
    visit new_contacts_path

    expect(page).to have_content I18n.t('contacts.contact_us')
  end

  scenario "allows a guest to create contact" do
    visit new_contacts_path
    fill_in :contact_email, with: 'foo@bar.ru'
    fill_in :contact_message, with: 'Foo Bar Baz'

    click_button 'Send message'
    expect(page).to have_content 'Thanks!'
  end

end
```

Следующий тест: протестировать функциональность приложения залогинившись под пользователем

Сделаем сначала **тест для гостя, что он может зарегистрироваться на сайте, т.е. протестируем форму регистрации.**

Создадим файл /spec/features/visitor_creates_account_spec.rb

```
require "rails_helper"

feature "Account Creation" do
  scenario "allows guest to create account" do
    visit new_user_registration_path

    fill_in :user_username, with: 'FooBar'
    fill_in :user_email, with: 'foo@bar.com'
    fill_in :user_password, with: '1234567'
    fill_in :user_password_confirmation, with: '1234567'

    click_button 'Sign up'
    expect(page).to have_content I18n.t('devise.registrations.signed_up')
  end
end
```

devise.registrations.signed_up взято из i18n - config/locales/devise.en.yml

Запустим тест:

```
rake spec
```

Всё это работает с базой данных test.sqlite3

Далее проверим функциональность создания статей зарегистрированным пользователем:

Чтобы не зависеть от порядка исполнения тестов и не повотояться в коде, вынесем часть кода в метод sign_up

/spec/features/visitor_creates_account_spec.rb:

```
require "rails_helper"

feature "Account Creation" do
  scenario "allows guest to create account" do
    sign_up
    expect(page).to have_content I18n.t('devise.registrations.signed_up')
  end
end

def sign_up
  visit new_user_registration_path

  fill_in :user_username, with: 'FooBar'
  fill_in :user_email, with: 'foo@bar.com'
  fill_in :user_password, with: '1234567'
  fill_in :user_password_confirmation, with: '1234567'

  click_button 'Sign up'
end
```

Далее вынесем код метода sign_up в файл в каталоге /spec/support

RSpec: before, after hooks

> https://relishapp.com/rspec/rspec-core/v/3-8/docs/hooks/before-and-after-hooks

Нам надо использовать sign_up в разных тестах, и чтобы не повторяться и не писать один и тот же код, мы используем before, after hooks

Исполняется перед каждым тестом в feature или describe:

```
before(:each) do
end
```

Исполняется перед всеми тестами в feature или describe:

```
before(:all) do
end
```

Перепишем тест /spec/features/visitor_creates_account_spec.rb:

```
require "rails_helper"

feature "Account Creation" do
  scenario "allows guest to create account" do
    sign_up
    expect(page).to have_content I18n.t('devise.registrations.signed_up')
  end
end
```

Мы вынесли код метода sign_up в файл /spec/support/session_helper.rb:

```
def sign_up
  visit new_user_registration_path

  fill_in :user_username, with: 'FooBar'
  fill_in :user_email, with: 'foo@bar.com'
  fill_in :user_password, with: '1234567'
  fill_in :user_password_confirmation, with: '1234567'

  click_button 'Sign up'
end
```

Создадим тест для проверки создания статьи залогиненым пользователем /spec/features/user_creates_article_spec.rb:

```

```

Решение вопроса с тем, что при выполнении тестов в нескольких тестах вызывается sign_up (и выпадает ошибка о том, что этот этот пользователь уже есть). БД создаётся перед тестами.

> https://github.com/teamcapybara/capybara#transactions-and-database-setup
>
> https://github.com/DatabaseCleaner/database_cleaner#rspec-example

Добавить в Gemfile:

```
group :test do
  gem 'database_cleaner'
end
```

```
bundle install
```

Создадим файл конфигурации database_cleaner в файле /spec/support/database_cleaner.rb:

```
RSpec.configure do |config|

  config.before(:suite) do
    DatabaseCleaner.strategy = :transaction
    DatabaseCleaner.clean_with(:truncation)
  end

  config.around(:each) do |example|
    DatabaseCleaner.cleaning do
      example.run
    end
  end

end
```

> Изучить документацию Capybara: - https://github.com/teamcapybara/capybara - https://www.rubydoc.info/github/teamcapybara/capybara/master

Урок 48

Скаффолдинг

```
rails new toy_app
bundle install
bundle update
```

Сделаем мини-приложение микропостинг:

```
---------------------
|  users            |
---------------------
id     integer
name   string
email  string
---------------------

---------------------
|  microposts       |
---------------------
id      integer
content text
user_id integer
---------------------
```

```
rails g scaffold User name:string email:string
```

Создаст одновременно модель, контроллер, экшены, представления, тесты.

```
bundle exec rake db:migrate
```

```
rails g scaffold Micropost content:text user_id:integer
```

Валидация Micropost (в модели):

```
validates :content, length: { maximum: 140 }, presence: true
validates :user_id, presence: true
```

Валидация User (в модели):

```
validates :name, presence: true
validates :email, presence: true
```

Добавим ассоциации one-to-many между пользователем и микропостом.

```
User --1--------*-- Micropost
```

Модель User:

```
has_many :microposts
```

Модель Micropost:

```
belongs_to :user
```

Откроем rails console:

```
first_user = User.first
first_user.microposts
first_user.microposts.count # обратится к БД
first_user.microposts.length # прочитает то, что уже есть в памяти, без БД
micropost = first_user.microposts.first # первый микропост первого пользователя
micropost.user # получить пользователя микропоста
```

- Модели наследуются от ActiveRecord::Base
- Контроллеры наследуются от ApplicationController, а ApplicationController от ActionController::Base

Отправка e-mail

- SMTP-сервер
- хостинг - большой процент попадания в спам
- gmail - надо включить options, есть ограничения
- postmarkapp - для Transactional emails, но через него нельзя делать почтовую рассылку. Письма должны быть с кнопкой Unsubscribe.
- bulk email messaging - для рассылки

> https://postmarkapp.com/developer/user-guide/sending-email/sending-with-api
>
> https://github.com/wildbit/postmark-gem
>
> https://github.com/wildbit/postmark-rails
>
> http://rusrails.ru/action-mailer-basics
>
> https://www.youtube.com/watch?v=FNOhpAWbiKA
>
> https://github.com/mikel/mail

Урок 49

Про паттерны программирования. Полиморфные ассоциации:

Создадим приложение poly_demo

```
rails new poly_demo
```

При написании приложения у нас могут быть сущности, которые содержит одинаковое поведение, одинаковые свойства и нам захочется использовать DRY-принцип, и в этом нам поможет следующее:

```
Post - PostComment (x)

Image - ImageComment (x)

Link - LinkComment (x)

Article - ArticleComment (x)
```

Нам нет смысла создавать отдельные сущности для подвидов комментариев.

```
Post
          \
Image    --  Comment
          /
Link

Article


Post.comments
Image.comments
Link.comments
Article.comments


content
rating
```

Одна модель может принадлежать разным сущностям, но при этом оставаться сама собой (полиморфизм).

```
rails g model Comment content:text
```

```
rails g model Post content:text
```

```
rails g model Image url:text
```

Свяжем эти сущности.

Т.к. нам надо связать сущность комментарий с несколькими сущностями, belongs_to делаетс немного иначе, чем обычно.

При связывании с полиморфной ассоциацией надо в belongs_to добавить с окончанием **able**

/app/models/comment.rb:

```
class Comment < ApplicationRecord
  belongs_to :commentable, polymorphic: true
end
```

commentable - получается: комментируемый

Хендл, рукоятка, которая существует у других сущностей.

/app/models/post.rb:

```
class Post < ApplicationRecord
  has_many :comments, as: :commentable
end
```

/app/models/image.rb:

```
class Image < ApplicationRecord
  has_many :comments, as: :commentable
end
```

Далее, откроем миграцию db/migrate/20190205095251_create_comments.rb и добавим строку:

```
t.references :commentable, polymorphic: true
```

```
class CreateComments < ActiveRecord::Migration[5.2]
  def change
    create_table :comments do |t|
      t.text :content
      t.references :commentable, polymorphic: true
      t.timestamps
    end
  end
end
```

Далее, делаем:

```
bundle exec rake db:migrate
```

Далее, откроем Rails-консоль:

```
rails console
```

```
post = Post.create(content: 'Foo bar')
post.comments
post.comments.create(content: 'Baz Buuu Foo')
post.comments.create(content: 'Comment 2')
post.comments
image = Image.create(url: '1.jpg')
image.comments.create(content: 'Wow! Super!')
image.comments.create(content: 'This is comment for image!')
image.comments
image2 = Image.create(url: '2.jpg')
image2.comments.create(content: 'Bar')
```

Посмотрим базу данных /db/development.sqlite3:

```
sqlite3 development.sqlite3
```

```
select * from comments;
select * from posts;
select * from images;
```

Про развёртывание на Heroku

> См. инструкцию тут: https://github.com/krdprog/prog-conspects/blob/master/hartl-rails.md

Паттерн Singleton (один объект на всех)

Банда Четырёх - GoF (Gang of Four)

> https://habr.com/ru/post/210288/
>
> https://ru.wikipedia.org/wiki/Design_Patterns

Напишем логгер, который будет сначала выводить на экран, а затем сохранять в файл.

app.rb:

```
class Logger
  def say_foo
    puts "Fooo!"
  end
end

logger = Logger.new
logger.say_foo
```

Создадим ещё логгеры. Есть недостаток такого подхода. В этом случае, объектов создаётся много, а действие совершается одно.

Попробуем исправить:

app.rb:

```
class Logger
  def self.say_foo
    puts "Fooo!"
  end
end

Logger.say_foo
```

Добавим ещё метод:

```
class Logger
  def self.say_foo
    puts "Fooo!"
  end

  def self.log_foo bar
    f = File.open 'log.txt', 'a'
    f.puts bar
    f.close
  end
end

Logger.say_foo
Logger.log_foo 'Wow!'
Logger.log_foo 'Meow!'
```

Недостаток в том, что мы постоянно открываем и закрываем файл log.txt - это увеличивает нагрузку.

class method

```
Logger.say_foo)
```

instance method

```
logger = Logger.new
logger.say_foo
```

Чтобы решить это, нам потребуется паттерн Singleton

Наша цель сделать один объект на всех.

Всё это принадлежит к экземпляру класса (к объекту):

@foo - instance variables

instance method:

```
def foo
end
```

Это принадлежит к статическому классу:

@@foo - class variable

class method:

```
def self.foo
end
```

Решение:

logger.rb:

```
class Loggeer
  def initialize
    @f = File.open 'log.txt', 'a'
  end

  # сделаем, чтобы метод возвращал экземпляр нашего класса
  @@x = Loggeer.new

  def self.instance
    return @@x
  end

  # instance method
  def log bar
    @f.puts bar
    @f.flush
  end

  # механизм защиты, чтобы Loggeer.new можно было написать только внутри класса
  private_class_method :new
end
```

app.rb:

```
require "./logger"

Loggeer.instance.log "Good job!"
```

В стандартной библиотеке ruby есть модуль Singleton

> https://ruby-doc.org/stdlib-2.5.3/libdoc/singleton/rdoc/Singleton.html

Перепишем программу используя этот встроенный модуль.

logger.rb:

```
require 'singleton'

class Loggeer

  include Singleton

  def initialize
    @f = File.open 'log.txt', 'a'
  end

  def log bar
    @f.puts bar
    @f.flush
  end

end
```

app.rb:

```
require "./logger"

Loggeer.instance.log "It`s work!"
```

> https://github.com/krdprog/logger.rb

Домашнее задание: сделать блог с сущностями post, link, image с комменатриями, сделать на главной странице вывод всех этих сущностей.

Урок 50

Регулярные выражения - Regular expression (regex)

> https://rubular.com/

Для отработки текстовых файлов, данных, веб-страниц.

Файл для тренеровки с Regex можно скачать тут: https://github.com/krdprog/rubyschool-notes/blob/master/for-lesson-50-regex.txt

```
The cat goes catatonic when you put in in catapult
```

```
cat\b

\b - граница слова

Выберет: cat
```

```
Yesterday, at 12 AM, he withdrew $600.00 from an ATM. He then spent $200.12 on  groceries. At 3:00 P.M., he logged onto a poker site

He had only $0.1 in his pocket.
```

```
\.\d{2}


\. - точка
\d - любая цифра
{2} - количество повторяющихся символов

Выберет: .00 .12 .41 .82

\.\d{1,2}

Выберет: .00 .12 .41 .82 0.1
```

```
hi world, it's hip to have big thighs
```

```
hi\b

Выберет: hi
```

```
this is file.txt

\b заменить на _

Результат: this_is_file.txt
```

Regex нужны для:

- проверки
- замены

В Sublime Text нажмём Ctrl + H (поиск и замена), и выберем "с Regular Expression"

Проблемы на пустом месте

Пробел, табуляция, \n, \r - всё это whitespace

```
\n\n
\n  \n
```

```
the quick brown

fox jumps over

the lazy dog
```

```
Убрать пустые строки:

\n\n заменяем на \n
```

```


the quick

brown fox



jumps


over

the lazy



dog
```

```
\n+ заменяем на \n
```

Шпаргалка по регулярным выражениям (regex)

```
[abc]  A single character of: a, b, or c
[^abc]  Any single character except: a, b, or c
[a-z]   Any single character in the range a-z
[a-zA-Z]  Any single character in the range a-z or A-Z
^   Start of line
$   End of line
\A  Start of string
\z  End of string
.   Any single character
\s  Any whitespace character
\S  Any non-whitespace character
\d  Any digit
\D  Any non-digit
\w  Any word character (letter, number, underscore)
\W  Any non-word character
\b  Any word boundary
(...)   Capture everything enclosed
(a|b)   a or b
a?  Zero or one of a
a*  Zero or more of a
a+  One or more of a
a{3}  Exactly 3 of a
a{3,}   3 or more of a
a{3,6}  Between 3 and 6 of a
```

```
\ - с обратной косой черты начинаются буквенные спецсимволы, а также он используется
если нужно использовать спецсимвол в виде какого-либо знака препинания;
^ - указывает на начало строки;
$ - указывает на конец строки;
* - указывает, что предыдущий символ может повторяться 0 или больше раз;
+ - указывает, что предыдущий символ должен повторится больше один или больше раз;
? - предыдущий символ может встречаться ноль или один раз;
{n} - указывает сколько раз (n) нужно повторить предыдущий символ;
{N,n} - предыдущий символ может повторяться от N до n раз;
. - любой символ кроме перевода строки;
[az] - любой символ, указанный в скобках;
x|y - символ x или символ y;
[^az] - любой символ, кроме тех, что указаны в скобках;
[a-z] - любой символ из указанного диапазона;
[^a-z] - любой символ, которого нет в диапазоне;
\b - обозначает границу слова с пробелом;
\B - обозначает что символ должен быть внутри слова, например, ux совпадет с uxb
или tuxedo, но не совпадет с Linux;
\d - означает, что символ - цифра;
\D - нецифровой символ;
\n - символ перевода строки;
\s - один из символов пробела, пробел, табуляция и так далее;
\S - любой символ кроме пробела;
\t - символ табуляции;
\v - символ вертикальной табуляции;
\w - любой буквенный символ, включая подчеркивание;
\W - любой буквенный символ, кроме подчеркивания;
\uXXX - символ Unicdoe.
```

Содержание конспекта: