Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»

(национальный исследовательский университет)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

О т ч е т

по лабораторной работе № 10

Дисциплина: Языки Интернет-программирования.

Студент гр. ИУ6-33Б _____ **Дасов Т.Д.**

(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____

(Подпись, дата) (И.О. Фамилия)

Москва, 2019

Задание.

Модифицировать код ЛР 8 таким образом, чтобы по запросу с указанными параметрами выдавался результат в формате XML (средствами стандартной сериализации ActiveSupport).

- Проверить формирование XML и сохранить в файл для отладки XSLT и второго приложения.
- Написать функциональный тест, проверяющий формат выдаваемых данных при запросе RSS.

Разработать XSLT-программу преобразования полученной XML в HTML.
Добавить в проверяемый XML-файл строку привязки к преобразованию <?xml-stylesheet type="text/xsl" href="some_transformer.xslt"?>. Проверить корректность отображения браузером результата преобразования.
Проверить на автономной Ruby-программе корректность преобразования, используя следующий фрагмент кода:

```
require 'nokogiri'
doc = Nokogiri::XML(File.read('some_file.xml'))
xslt = Nokogiri::XSLT(File.read('some_transformer.xslt'))
puts xslt.transform(doc)
```

Разработать второе приложение, являющееся посредником между клиентом и первым приложением, задачей которого является преобразование XML в HTML или передача в неизменном виде браузеру для отображения браузером. Приложение должно запускаться с указанием номера порта TCP, отличным от номера порта первого приложения (например rails server -p 3001)!

- Подготовить каркас приложения, а также форму формирования запроса, форму отображения результата и соответствующие действия контролера.
- Добавить в контроллер преобразование XML в HTML с помощью ранее разработанного XSLT-файла.
- Подключить запрос XML с первого приложения и проверить работу приложений в связке.
- Написать функциональный тест, проверяющий что при различных входных данных результат генерируемой страницы различен.
- Доработать код контроллера и представлений данного приложения для выдачи браузеру XML-потока в неизменном виде (организовать возможность выбора формата выдачи для пользователя).
- Проверить, что браузер получает XML первого приложения в неизменном виде.
- Доработать код контроллера приложения таким образом, чтобы XML-поток первого приложения получал дополнительную строку, указывающую xsl. Модифицировать форму запроса параметров таким образом, чтобы браузер получал в ответ XML. При этом разместить XSLT-файл в директории public.
- Проверить, что браузер производит преобразование XML->HTML в соответствии с xlt.
- Реализовать функциональные тесты второго приложения. Проверить результаты, формируемые приложением, на соответствие выбранному формату выдачи.

Итоговая форма ввода параметра должна содержать кнопки или селектор, позволяющие проверить два варианта преобразования:

- Серверное xml+xslt->html
- Клиентское xml+xslt->html

Маршруты routes.rb:

```
Rails.application.routes.draw do
  root 'main#index'
  get 'main/result' => 'main#result', :as => 'result'
  get 'main/index' => 'main#index', :as => 'index'
end
```

Изменения внесенные в контроллер основного приложения:

```
# frozen_string_literal: true

# MainController class
class MainController < ApplicationController
  def index; end

  def result
    arr = []
    par = params[:n].to_i
    par.times do |i|
      par.times do |j|
        arr << [i, j] if compare_nums(i, j) && !arr.include?([j, i]) && i != j
      end
    end
    render xml: arr.to_xml
  end

  private

  def compare_nums(a_val, b_val)
    find_dev(a_val).reduce(:+) == b_val && find_dev(b_val).reduce(:+) == a_val
  end

  def find_dev(x_val)
    res = []
    devider = x_val / 2
    while devider.positive?
      res.push(devider) if (x_val % devider).zero?
      devider -= 1
    end
    res
  end
end
```

Тест первого приложения (сравнение разных запросов):

```
# frozen_string_literal: true

require 'test_helper'

# Test should compare results of different requests
class CompareResponceTest < ActionDispatch::IntegrationTest
  def setup; end

  def teardown; end

  test 'compare responces' do
    get result_path, params: { n: 300 }
    res1 = response.body
    get result_path, params: { n: 1300 }
    res2 = response.body
    assert_not_equal res1, res2
  end

  test 'rss request' do
    get result_url, params: { n: '300', format: :rss }
    assert response.body.include?('<?xml version="')
  end
end
```

Контроллер proxy сервера (main_controller.rb):

```
# frozen_string_literal: true

require 'nokogiri'
require 'net/http'

# MainController class
class MainController < ApplicationController
  XSLT_PUBLIC = "#{Rails.root}/public/parser.xslt"
  SERVER_URL = 'http://localhost:3000/main/result.xml'

  def index; end

  def result
    res = Nokogiri::XML(Net::HTTP.get(URI("#{SERVER_URL}?n=#{params[:n]}")))
    case params[:handle]
    when 'Сервер'
      render inline: XmlConverter.call(res, XSLT_PUBLIC).to_html
    when 'Браузер'
      render xml: XsltAppender.call(res, '/parser.xslt').to_xml
    when 'Не обрабатывать'
      render xml: res
    end
  end
end
```

Сервисный объект для преобразования XML в HTML (xml_converter.rb):

```
# frozen_string_literal: true

require 'nokogiri'

# Class for handling responses
class XmlConverter < Service
  include Dry::Monads[:result, :do]

  def call(xml, href)
    yield convert(xml, href)
  end

  private

  def convert(xml_doc, xslt_href)
    xslt = Nokogiri::XSLT(File.read(xslt_href))
    result = xslt.transform(xml_doc)
    return Success(result) unless result.nil?

    Failure(:error_while_converting)
  end
end
```

Сервисный объект для добавления инструкции XSLT (xslt_appender.rb):

```
# frozen_string_literal: true

require 'nokogiri'

# Appends xslt instruction
class XsltAppender < Service
  include Dry::Monads[:result, :do]

  def call(xml, xslt_path)
    yield append(xml, xslt_path)
  end

  private

  def append(xml_doc, ref)
    instr = Nokogiri::XML::ProcessingInstruction
          .new(xml_doc, 'xml-stylesheet', "type=\"text/xsl\" href=\"#{ref}\"")
    xml_doc.root.add_previous_sibling instr
    return Success(xml_doc) unless xml_doc.nil?

    Failure(:error_while_appending_instr)
  end
end
```

Родительский класс для сервисных объектов (service.rb):

```
# frozen_string_literal: true

# Parent class for all services
class Service
  class << self
    def call(*args, &block)
      new.call(*args, &block)
    end
  end
end
```

XSLT программа (parser.xslt):

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/arrays">
      <html>
      <head>
         <style>
            .table-elem {
               text-align: center;
               background-color: #EEEEEE;
            }

            .table-elem:nth-of-type(even) {
               background-color: #DDDDDD;
            }

            .table-elem td {
               padding: 15px;
            }
```

```
        </style>
      </head>
      <body>
        <table>
          <th colspan="2">
            <td>Дружественные числа в заданном диапазоне</td>
          </th>
          <xsl:for-each select="array">
            <tr class="table-elem">
              <td><xsl:value-of select="array[1]"/></td>
              <td><xsl:value-of select="array[2]"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
      </html>
  </xsl:template>
</xsl:stylesheet>
```

Тест для проверки формата ответа сервера (check_format_test.rb):

```
# frozen_string_literal: true

require 'test_helper'

class CheckFormat < ActionDispatch::IntegrationTest
  def setup; end

  def teardown; end

  test 'check responce with browser processing' do
    get result_path, params: { n: 300, handle: 'Браузер' }
    content = '<?xml-stylesheet type="text/xsl" href="/parser.xslt"?>'
    assert response.body.include?(content)
  end

  test 'check response with server processing' do
    get result_path, params: { n: 300, handle: 'Сервер' }
    content = ['<html>', '<head>', '<body>', '<table', '<th', '<tr']
    res = response.body
    assert(content.all? { |str| res.include?(str) })
  end

  test 'check response without any handlings' do
    get result_path, params: { n: 300, handle: 'Не обрабатывать' }
    content = ['<?xml version="1.0" encoding="UTF-8"?>', '<arrays type="array">']
    not_incl = '<?xml-stylesheet type="text/xsl" href="/parser.xslt"?>'
    res = response.body
    assert(content.all? { |str| res.include?(str) } && res.exclude?(not_incl))
  end
end
```

Пример работы приложения:

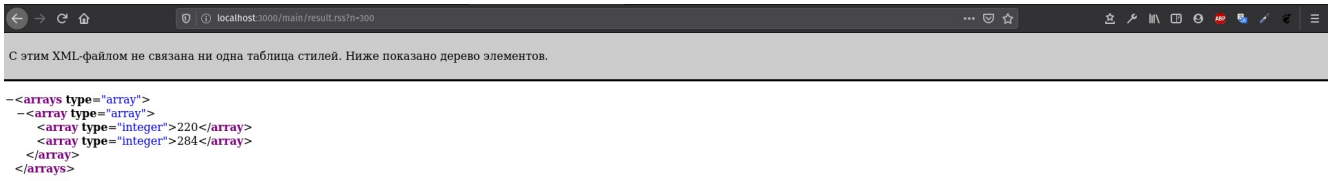

Рис. 1 (XML ответ основного сервера)

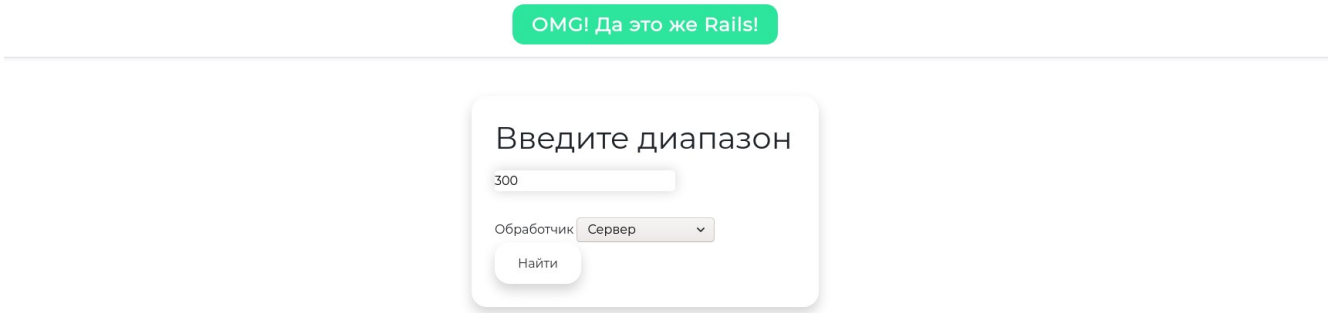

Рис. 2 (Главная страница proxy сервера)

Дружественные числа в заданном диапазоне	
220	284

Рис. 3 (XML обработан на стороне клиента)

Тесты основного приложения:

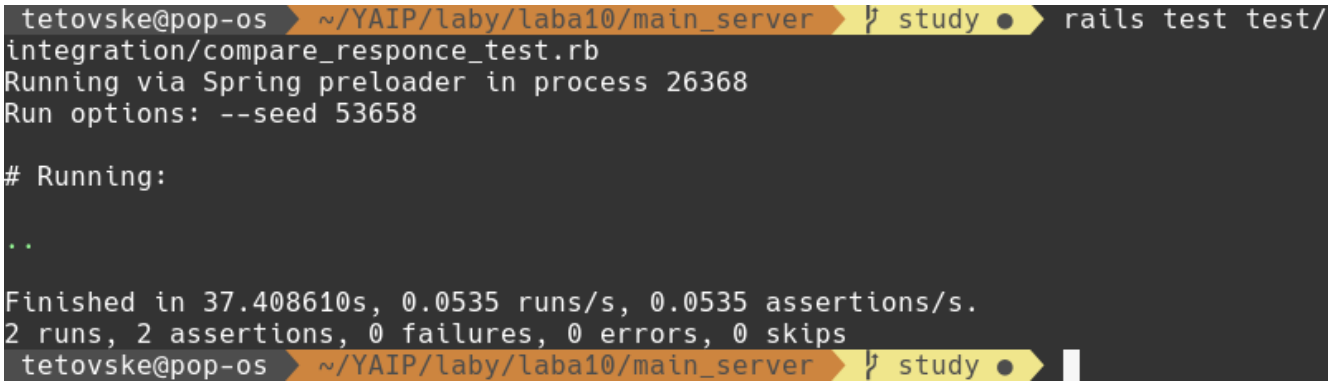

Рис. 4 (Сравнение ответов на разные запросы)

Тесты proxy сервера:

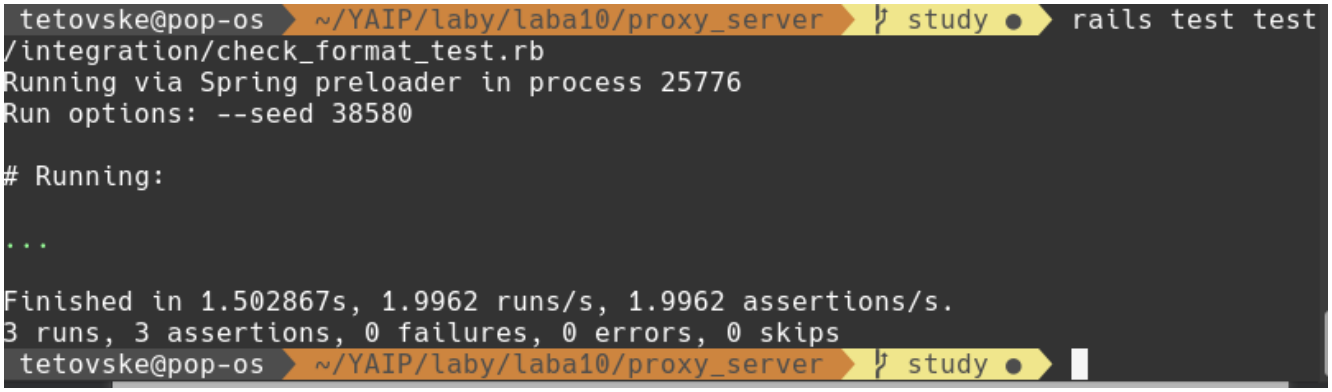

Рис. 5 (Проверка на разные форматы выдачи)

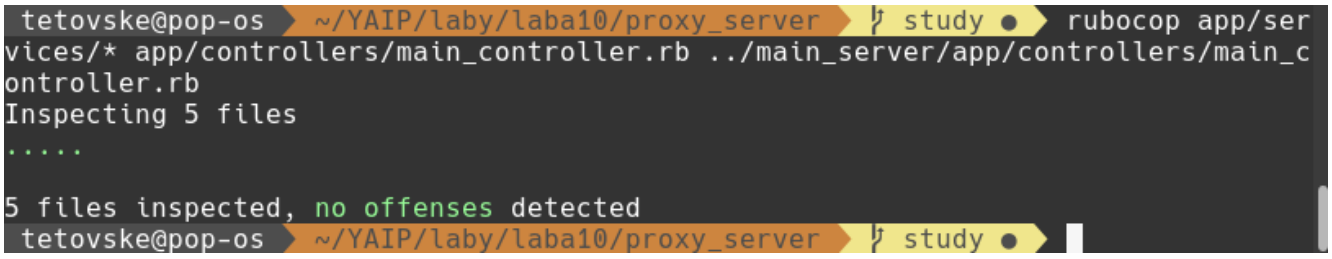

Рис. 6 (Отчет Rubocop)

Вывод: *в ходе выполнения лабораторной работы было разработано приложение-посредник для преобразования XML ответа основного сервера. Была разработана XSLT программа для преобразования XML в HTML. Приложение было протестировано и проверено на соответствие стилю программой Rubocop.*