



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 10

Название: Аjax

Дисциплина: Языки интернет-программирования

Студент

ИУ6-31Б

(Группа)

(Подпись, дата)

Лазарев Е.С.

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

Малахов Д.В.

(И.О. Фамилия)

Москва, 2022

Цель работы: изучить базовый принцип работы XSLT Ruby on Rails

ЛР 10. Формирование и отображение XML в HTML средствами сервера и клиента.

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 контроллера.
- 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`

Результаты должны быть представлены в виде двух файлов:

отчет в формате pdf/odt/doc;
архив rails-приложения (zip, tgz, 7z).

XSLT-API

XML Controller:

```
class XmlController < ApplicationController
  require 'prime'
  before_action :parse_params, only: :index
  def index
    mass, rez1 = [], []
    i=0
    mass = Prime.each(@value).map(&:to_i)
    a = 0
    rez2 = mass.map{|p| 2**p - 1}.select {|a| a <= mass.last && Prime.prime?(a) }
    rez1= rez2.map.with_index do |num, index|
      {iteration: index+1, value: num}
    end
    respond_to do |format|
      format.xml { render xml: rez1.to_xml }
      format.rss { render xml: rez1.to_xml }
    end
  end
  protected
  def parse_params
    @value = params[:val].to_i
  end
end
```

Тест:

```
require 'rails_helper'
require 'spec_helper'
```

```
RSpec.describe 'XmIs', type: :request do
```

```
  describe 'GET /' do
    it 'returns http success' do
      get '/', params: { val: 10, format: :rss }
```

```

expect(response).to have_http_status(:success)
expect(response.headers['Content-Type']).to include('application/rss+xml')
end

it 'Compares two responses with different values' do
  get '/', params: { val: rand(1..29), format: :xml }
  response1 = response
  get '/', params: { val: rand(30..101), format: :xml }

  expect(response.body).not_to eq(response1.body)
end
end
end

```

XSLT-PROXY

Controller:

```

require 'nokogiri'
require 'open-uri'
class ProxyController < ApplicationController
  before_action :parse_params, only: :output
  def input; end

  def output
    @val = params[:val]
    @side = params[:side]
    my_url = URL + "&val=#{@val}"
    api_response = URI.open(my_url)
    if @side == 'server'
      @result = xslt_transform(api_response).to_html
    elsif @side == 'client-with-xslt'
      render xml: insert_browser_xslt(api_response).to_xml
    else
      render xml: api_response
    end
  end

  private

  URL = 'http://localhost:3000/?format=xml'.freeze
  SERV_TRANS = "#{Rails.root}/public/server_transform.xslt".freeze
  BROWS_TRANS = '/browser_transform.xslt'.freeze
  def parse_params
    @val = params[:val]
    @side = params[:side]
  end

  def xslt_transform(data, transform: SERV_TRANS)
    doc = Nokogiri::XML(data)
    xslt = Nokogiri::XSLT(File.read(transform))
    xslt.transform(doc)
  end

  def insert_browser_xslt(data, transform: BROWS_TRANS)
    doc = Nokogiri::XML(data)
    xslt = Nokogiri::XML::ProcessingInstruction.new(doc, 'xml-stylesheet', 'type="text/xsl" href="" + transform + ""')
    doc.root.add_previous_sibling(xslt)
    doc
  end
end

```

Input:

```
<%= form_tag url_for(controller: :proxy, action: :output), method: 'get' do %>
<label>Введите натуральное число:</label>
<input type = "number" pattern = "^[\\d]+$" name = "val" id = "val"/>
<div>
<label>Обработать: </label>
<select class="custom-select" name = "side">
<option value = "server">Server XSLT</option>
<option value = "client-with-xslt">Client XSLT</option>
<option value = "client">Default XML</option>
</select>
</div>
<input type="submit" class = "btn btn-primary" value="Результат"/>
<% end %>
```

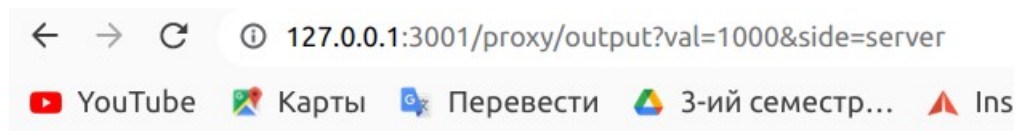
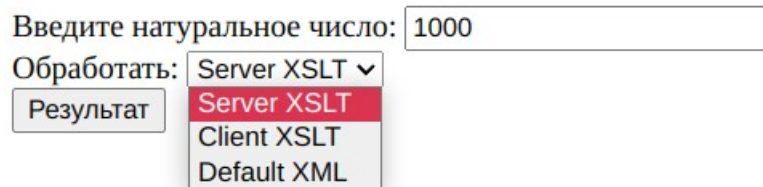
Output:

```
<h1>Ответ сервера</h1>
<%= render inline: @result %>
```

Test:

```
require 'rails_helper'
require 'spec_helper'
```

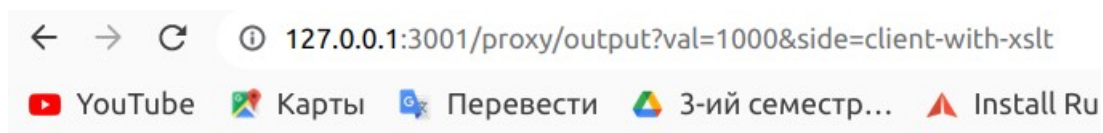
```
RSpec.describe 'Proxy', type: :request do
  describe 'GET /' do
    it 'Check server response' do
      { 'server': 'html', 'client': 'xml', 'client-with-xslt': 'xml' }.each do |side, type|
        get '/proxy/output', params: { val: rand(1..30), side: side, commit: 'Отправить' }
        expect(response).to have_http_status(:success)
        expect(response.headers['Content-Type']).to include(type)
      end
    end
  end
end
```



Ответ сервера

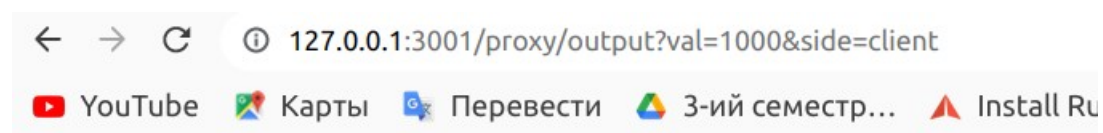
Номер Число Мерсена

1	3
2	7
3	31
4	127



Номер Число Мерсена

1	3
2	7
3	31
4	127



This XML file does not appear to have any style information associated with it. The

```
▼<objects type="array">
  <script/>
  <script/>
  ▼<object>
    <iteration type="integer">1</iteration>
    <value type="integer">3</value>
  </object>
  ▼<object>
    <iteration type="integer">2</iteration>
    <value type="integer">7</value>
  </object>
  ▼<object>
    <iteration type="integer">3</iteration>
    <value type="integer">31</value>
  </object>
  ▼<object>
    <iteration type="integer">4</iteration>
    <value type="integer">127</value>
  </object>
</objects>
```

Вывод: я изучил базовый принцип работы XSLT Ruby on Rails