Messaging
In this laboratory work I implemented messaging pattern to build a “number-processing” service. Which does the following tasks:
Get the number(s) and do some mathematical operations, they are “factorization”, “is prime”, “factorial”, “exponentiation”.
This project was written on Ruby on Rails. For working with message queuing I used RabbitMQ server. And Bunny gem, which is ruby client for working with RabbitMQ. Also I used Sneakers gem which makes easier creating workers, message handling and more things out of the box.
Let's analyza code:

This is our browser view, which sends messages to the HomeController#lab5

```
#lab5-1/app/views/home/about.html.erb
<div style="width: 400px; margin: 0 auto;">
      <select name="type" style="float: left">
            <option value="factorization">Factorization</option>
            <option value="is_prime">Is Prime</option>
            <option value="factorial">Factorial</option>
            <option value="exponentiation">Exponentiation</option>
      </select>
      <br><br>
      <input id="val1" name="value" placeholder="Enter your value" style="float:
left" type="number">
      <input id="hid" name="value1" placeholder="Enter second value"
style="float: left; visibility: hidden" type="number">
      <br><br><br>
      <button id="sub">Submit</button>
</div>
<script>
      $(document).ready(function(){
            $("select").change(function(){
                  var val = $(this).val()
                  if (val == 'exponentiation')
                  {
                        $("#hid").css('visibility', 'visible')
                  }
                  else
                  {
                        $("#hid").css('visibility', 'hidden')
                  }
            });
            $("#sub").click(function(){
                  var type = $("select").val();
                  var val1 = $("#val1").val();
                  var val2 = $("#hid").val();
                  $.ajax({
                    method: "POST",
                    url: "/lab5",
                    dataType: "json",
                    data: { type: type, value: val1, value1: val2 }
                  });
            });
      });
</script>
```

HomeController, which sends messages by method Publisher.publish

```
#lab5-1/app/controllers/home_controller.rb
class HomeController < ApplicationController

      def about

      end

  def lab5
    type = params[:type]
    value = params[:value]
    value1 = params[:value1]
    if type == 'exponentiation'
      value = value.to_s+" "+value1.to_s
    end

    Publisher.publish("calcs", type.to_s , value.to_s)

    @s = {}
    @s[:status] = value
    render json: @s
  end

end
```

Publisher class which initializes Bunny connection and send messages

```
class Publisher#lab5-1/app/services/publisher.rb
  # In order to publish message we need a exchange name.
  # Note that RabbitMQ does not care about the payload -
  # we will be using JSON-encoded strings
  def self.publish(exchange, message1 = '', message2 = '')
    # grab the fanout exchange
    x = channel.fanout("lab5.#{exchange}")
    puts "hehe"+message1.to_s+message2.to_s
    # and simply publish message
    x.publish(message1.to_s+"+"+message2.to_s)
  end

  def self.channel
    @channel ||= connection.create_channel
  end

  # We are using default settings here
  # The `Bunny.new(...)` is a place to
  # put any specific RabbitMQ settings
  # like host or port
  def self.connection
    @connection ||= Bunny.new.tap do |c|
      c.start
    end
  end
end
```

Setting up routing

```
# lab5-2/Rakefile
require 'sneakers/tasks'
require File.expand_path('../config/application', __FILE__)

Intrade::Application.load_tasks

# config/Rakefile
```

```
namespace :rabbitmq do
  desc "Setup routing"
  task :setup do
    require "bunny"

    conn = Bunny.new
    conn.start

    ch = conn.create_channel

    # get or create exchange
    x = ch.fanout("lab5.calcs")

    # get or create queue (note the durable setting)
    queue = ch.queue("dashboard.calcs", durable: true)

    # bind queue to exchange
    queue.bind("lab5.calcs")

    conn.close
  end
end
```

Here we setting up our Sneakers` configuration. There are running 4 workers by default.

```
#lab5-2/config/initializers/sneakers.rb
Sneakers.configure :timeout_job_after => 0
Sneakers.logger.level = Logger::INFO # the default DEBUG is too noisy
```

Worker class

```
#lab5-2/app/workers/calcs_worker.rb
class CalcsWorker
  include Sneakers::Worker
  # This worker will connect to "dashboard.posts" queue
  # env is set to nil since by default the actuall queue name would be
  # "dashboard.posts_development"
  from_queue "dashboard.calcs", env: nil

  # work method receives message payload in raw format
  # in our case it is JSON encoded string
  # which we can pass to RecentPosts service without
  # changes
  def work(raw_post)
    RecentCalcs.push(raw_post)
    ack! # we need to let queue know that message was received
  end
end
```

And finally RecentCalcs class which gets messages from queue by push method calculate our answer

```
#lab5-2/app/services/recent_calcs.rb
class RecentCalcs
  def self.push(raw_post)
      my_arr = raw_post.split('+')
      type = my_arr[0]
      value = my_arr[1]
      if type == "factorization"
            puts factorization(value)
      elsif type == "is_prime"
            puts is_prime(value)
  elsif type == "factorial"
    puts factorial(value)
      elsif type == "exponentiation"
            puts exponentiation(value)
```

```
      end
  end

  def self.factorization(value)
    require 'prime'
      @pd = (value.to_i).prime_division
      return "factorization of "+value.to_s+": "+@pd.to_s
  end

  def self.is_prime(value)
    require 'prime'
    if (value.to_i).prime?
           return value.to_s+" is prime"
      else
           return value.to_s+" is not prime"
      end
  end

  def self.factorial(value)
      f = 1; for i in 1..value.to_i; f *= i; end; f
      return "factorial of "+value.to_s+": "+f.to_s
  end

  def self.exponentiation(value)
      my_arr = value.split(' ')
      val1 = my_arr[0].to_i
      val2 = my_arr[1].to_i

      return "exponentiation of "+val1.to_s+" to degree "+val2.to_s+" is equal
to "+(val1**val2).to_s
  end

end
```

Finally we run lab5-1, and with other terminal type commands:

```
rake rabbitmq:setup
WORKERS=CalcsWorker rake sneakers:run
```