Rendering data as graphs

In this article

Learn how to visualize the programming languages from your repository using the D3.js library and Ruby Octokit.

In this guide, we're going to use the API to fetch information about repositories that we own, and the programming languages that make them up. Then, we'll visualize that information in a couple of different ways using the D3.js library. To interact with the GitHub API, we'll be using the excellent Ruby library, Octokit.

If you haven't already, you should read the "Basics of Authentication" guide before starting this example. You can find the complete source code for this project in the platform-samples repository.

Let's jump right in!

Setting up an OAuth app

First, register a new application on GitHub Enterprise Cloud. Set the main and callback URLs to `http://localhost:4567/`. As before, we're going to handle authentication for the API by implementing a Rack middleware using sinatra-auth-github:

```
require 'sinatra/auth/github'

module Example
  class MyGraphApp < Sinatra::Base
    # !!! DO NOT EVER USE HARD-CODED VALUES IN A REAL APP !!!
    # Instead, set and test environment variables, like below
    # if ENV['GITHUB_CLIENT_ID'] && ENV['GITHUB_CLIENT_SECRET']
    #  CLIENT_ID        = ENV['GITHUB_CLIENT_ID']
    #  CLIENT_SECRET    = ENV['GITHUB_CLIENT_SECRET']
    # end

    CLIENT_ID = ENV['GH_GRAPH_CLIENT_ID']
    CLIENT_SECRET = ENV['GH_GRAPH_SECRET_ID']

    enable :sessions

    set :github_options, {
      :scopes    => "repo",
      :secret    => CLIENT_SECRET,
      :client_id => CLIENT_ID,
      :callback_url => "/"
    }

    register Sinatra::Auth::Github

    get '/' do
```

```
      if !authenticated?
        authenticate!
      else
        access_token = github_user["token"]
      end
    end
  end
end
```

Set up a similar *config.ru* file as in the previous example:

```
ENV['RACK_ENV'] ||= 'development'
require "rubygems"
require "bundler/setup"

require File.expand_path(File.join(File.dirname(__FILE__), 'server'))

run Example::MyGraphApp
```

Fetching repository information

This time, in order to talk to the GitHub API, we're going to use the Octokit Ruby library. This is much easier than directly making a bunch of REST calls. Plus, Octokit was developed by a GitHubber, and is actively maintained, so you know it'll work.

Authentication with the API via Octokit is easy. Just pass your login and token to the `Octokit::Client` constructor:

```
if !authenticated?
  authenticate!
else
  octokit_client = Octokit::Client.new(:login => github_user.login, :oauth_token
=> github_user.token)
end
```

Let's do something interesting with the data about our repositories. We're going to see the different programming languages they use, and count which ones are used most often. To do that, we'll first need a list of our repositories from the API. With Octokit, that looks like this:

```
repos = client.repositories
```

Next, we'll iterate over each repository, and count the language that GitHub Enterprise Cloud associates with it:

```
language_obj = {}
repos.each do |repo|
  # sometimes language can be nil
  if repo.language
    if !language_obj[repo.language]
      language_obj[repo.language] = 1
    else
      language_obj[repo.language] += 1
    end
  end
end

languages.to_s
```

When you restart your server, your web page should display something that looks like

this:

```
{"JavaScript"=>13, "PHP"=>1, "Perl"=>1, "CoffeeScript"=>2, "Python"=>1,
"Java"=>3, "Ruby"=>3, "Go"=>1, "C++"=>1}
```

So far, so good, but not very human-friendly. A visualization would be great in helping us understand how these language counts are distributed. Let's feed our counts into D3 to get a neat bar graph representing the popularity of the languages we use.

Visualizing language counts

D3.js, or just D3, is a comprehensive library for creating many kinds of charts, graphs, and interactive visualizations. Using D3 in detail is beyond the scope of this guide, but for a good introductory article, check out "D3 for Mortals."

D3 is a JavaScript library, and likes working with data as arrays. So, let's convert our Ruby hash into a JSON array for use by JavaScript in the browser.

```
languages = []
language_obj.each do |lang, count|
  languages.push :language => lang, :count => count
end

erb :lang_freq, :locals => { :languages => languages.to_json}
```

We're simply iterating over each key-value pair in our object and pushing them into a new array. The reason we didn't do this earlier is because we didn't want to iterate over our `language_obj` object while we were creating it.

Now, *lang_freq.erb* is going to need some JavaScript to support rendering a bar graph. For now, you can just use the code provided here, and refer to the resources linked above if you want to learn more about how D3 works:

```
<!DOCTYPE html>
<meta charset="utf-8">
<html>
  <head>
    <script src="//cdnjs.cloudflare.com/ajax/libs/d3/3.0.1/d3.v3.min.js">
</script>
    <style>
    svg {
      padding: 20px;
    }
    rect {
      fill: #2d578b
    }
    text {
      fill: white;
    }
    text.yAxis {
      font-size: 12px;
      font-family: Helvetica, sans-serif;
      fill: black;
    }
    </style>
  </head>
  <body>
    <p>Check this sweet data out:</p>
    <div id="lang_freq"></div>

  </body>
  <script>
    var data = <%= languages %>;
```

```
    var barWidth = 40;
    var width = (barWidth + 10) * data.length;
    var height = 300;

    var x = d3.scale.linear().domain([0, data.length]).range([0, width]);
    var y = d3.scale.linear().domain([0, d3.max(data, function(datum) { return
datum.count; })]).
      rangeRound([0, height]);

    // add the canvas to the DOM
    var languageBars = d3.select("#lang_freq").
      append("svg:svg").
      attr("width", width).
      attr("height", height);

    languageBars.selectAll("rect").
      data(data).
      enter().
      append("svg:rect").
      attr("x", function(datum, index) { return x(index); }).
      attr("y", function(datum) { return height - y(datum.count); }).
      attr("height", function(datum) { return y(datum.count); }).
      attr("width", barWidth);

    languageBars.selectAll("text").
      data(data).
      enter().
      append("svg:text").
      attr("x", function(datum, index) { return x(index) + barWidth; }).
      attr("y", function(datum) { return height - y(datum.count); }).
      attr("dx", -barWidth/2).
      attr("dy", "1.2em").
      attr("text-anchor", "middle").
      text(function(datum) { return datum.count;});

    languageBars.selectAll("text.yAxis").
      data(data).
      enter().append("svg:text").
      attr("x", function(datum, index) { return x(index) + barWidth; }).
      attr("y", height).
      attr("dx", -barWidth/2).
      attr("text-anchor", "middle").
      text(function(datum) { return datum.language;}).
      attr("transform", "translate(0, 18)").
      attr("class", "yAxis");
  </script>
</html>
```

Phew! Again, don't worry about what most of this code is doing. The relevant part here is a line way at the top--`var data = <%= languages %>;`--which indicates that we're passing our previously created `languages` array into ERB for manipulation.

As the "D3 for Mortals" guide suggests, this isn't necessarily the best use of D3. But it does serve to illustrate how you can use the library, along with Octokit, to make some really amazing things.

Combining different API calls

Now it's time for a confession: the `language` attribute within repositories only identifies the "primary" language defined. That means that if you have a repository that combines several languages, the one with the most bytes of code is considered to be the primary language.

Let's combine a few API calls to get a *true* representation of which language has the greatest number of bytes written across all our code. A treemap should be a great way

to visualize the sizes of our coding languages used, rather than simply the count. We'll need to construct an array of objects that looks something like this:

```
[ { "name": "language1", "size": 100},
  { "name": "language2", "size": 23}
  ...
]
```

Since we already have a list of repositories above, let's inspect each one, and call the GET /repos/{owner}/{repo}/languages endpoint:

```
repos.each do |repo|
  repo_name = repo.name
  repo_langs = octokit_client.languages("#{github_user.login}/#{repo_name}")
end
```

From there, we'll cumulatively add each language found to a list of languages:

```
repo_langs.each do |lang, count|
  if !language_obj[lang]
    language_obj[lang] = count
  else
    language_obj[lang] += count
  end
end
```

After that, we'll format the contents into a structure that D3 understands:

```
language_obj.each do |lang, count|
  language_byte_count.push :name => "#{lang} (#{count})", :count => count
end

# some mandatory formatting for D3
language_bytes = [ :name => "language_bytes", :elements => language_byte_count]
```

(For more information on D3 tree map magic, check out this simple tutorial.)

To wrap up, we pass this JSON information over to the same ERB template:

```
erb :lang_freq, :locals => { :languages => languages.to_json,
:language_byte_count => language_bytes.to_json}
```

Like before, here's a bunch of JavaScript that you can drop directly into your template:

```
<div id="byte_freq"></div>
<script>
  var language_bytes = <%= language_byte_count %>
  var childrenFunction = function(d){return d.elements};
  var sizeFunction = function(d){return d.count;};
  var colorFunction = function(d){return Math.floor(Math.random()*20)};
  var nameFunction = function(d){return d.name;};

  var color = d3.scale.linear()
              .domain([0,10,15,20])
              .range(["grey","green","yellow","red"]);

  drawTreemap(5000, 2000, '#byte_freq', language_bytes, childrenFunction,
nameFunction, sizeFunction, colorFunction, color);

  function
drawTreemap(height,width,elementSelector,language_bytes,childrenFunction,nameFunctio
{
```

```
      var treemap = d3.layout.treemap()
          .children(childrenFunction)
          .size([width,height])
          .value(sizeFunction);

      var div = d3.select(elementSelector)
          .append("div")
          .style("position","relative")
          .style("width",width + "px")
          .style("height",height + "px");

      div.data(language_bytes).selectAll("div")
          .data(function(d){return treemap.nodes(d);})
          .enter()
          .append("div")
          .attr("class","cell")
          .style("background",function(d){ return colorScale(colorFunction(d));})
          .call(cell)
          .text(nameFunction);
  }

  function cell(){
      this
          .style("left",function(d){return d.x + "px";})
          .style("top",function(d){return d.y + "px";})
          .style("width",function(d){return d.dx - 1 + "px";})
          .style("height",function(d){return d.dy - 1 + "px";});
  }
</script>
```

Et voila! Beautiful rectangles containing your repo languages, with relative proportions that are easy to see at a glance. You might need to tweak the height and width of your treemap, passed as the first two arguments to `drawTreemap` above, to get all the information to show up properly.