How to build a dynamic web application using PSGI

PSGI | Plack | plackup

Prev Next

Now that we have built our first web application using PSGI we can go a step further and build something that can respond to a query.

In order to build a web application that can accept input from users we need to understand how Plack/PSGI defined that part of the interaction.

Back in the CGI world we had a mix of environment variables in action. In PSGI everything interesting will be passed via a single parameter to the anonymous subroutine that implements our application. That parameter is a reference to a HASH. Let's see what does it contain:

```
 1. #!/usr/bin/perl
    use strict;
    use warnings;

 5. use Data::Dumper qw(Dumper);
    $Data::Dumper::Sortkeys = 1;

    my $app = sub {
      my $env = shift;
10.   return [
11.     '200',
        [ 'Content-Type' => 'text/plain' ],
        [ Dumper $env ],
      ];
15. };
```

Save this in a file called `env.psgi` and run it using `plackup env.psgi`. When it is launched visit the http://localhost:5000/ with your favorite browser. You will see something like this:

```
$VAR1 = {
    'HTTP_ACCEPT' => 'text/html,application/xhtml+xml,...',
    'HTTP_ACCEPT_CHARSET' => 'ISO-8859-1,utf-8;q=0.7,*;q=0.7',
    'HTTP_ACCEPT_ENCODING' => 'gzip, deflate',
    'HTTP_ACCEPT_LANGUAGE' => 'en-gb,en;q=0.5',
    'HTTP_CACHE_CONTROL' => 'max-age=0',
    'HTTP_CONNECTION' => 'keep-alive',
```

```
    'HTTP_COOKIE' => '__utma=1118.128.1348.1379.107.6; __utmz=111.1348.1.1....',
    'HTTP_HOST' => 'localhost:5000',
    'HTTP_USER_AGENT' => 'Mozilla/5.0 (Windows NT 6.1; rv:9.0.1) ...',
    'PATH_INFO' => '/',
    'QUERY_STRING' => '',
    'REMOTE_ADDR' => '127.0.0.1',
    'REQUEST_METHOD' => 'GET',
    'REQUEST_URI' => '/',
    'SCRIPT_NAME' => '',
    'SERVER_NAME' => 0,
    'SERVER_PORT' => 5000,
    'SERVER_PROTOCOL' => 'HTTP/1.1',
    'psgi.errors' => *::STDERR,
    'psgi.input' => \*{'HTTP::Server::PSGI::$input'},
    'psgi.multiprocess' => '',
    'psgi.multithread' => '',
    'psgi.nonblocking' => '',
    'psgi.run_once' => '',
    'psgi.streaming' => 1,
    'psgi.url_scheme' => 'http',
    'psgi.version' => [
                       1,
                       1
                      ],
    'psgix.input.buffered' => 1,
    'psgix.io' => bless( \*Symbol::GEN1, 'IO::Socket::INET' )
};
```

Let's see what do we have here.

In the script we used the `Dumper` function of the standard Data::Dumper module. By default it would print out the data without any order. Setting the `$Data::Dumper::Sortkeys` variable to 1 changes that behavior and the hash keys are sorted. That makes it a lot easier to read.

We also set the **Content-Type** to be **text/plain**. HTML normally disregards spaces so with this we are telling the visiting browser to interpret our data as plain text. That way it will display the data verbatim. Keeping the spaces and the newlines.

The data itself can be divided into two parts. The first part - a set of upper case keys - are the familiar set of environment variables. The second part is a set of PSGI specific keys. I won't go over any of these, they are described in the PSGI specification.

A GET request with parameters

If you are familiar with HTTP then you might want to know how does a GET request with some parameters show up in this raw data.

Let's change our request to the following: `http://localhost:5000/page?name=value`

In the data we can see the following changes:

```
  'PATH_INFO' => '/page',
  'QUERY_STRING' => 'name=value',
  'REQUEST_URI' => '/page?name=value',
```

Simple echo server

In order to go a small step further, we are going to build a simple echo server. This is a page with a single entry field and a button. When you press the button it will reload itself and display the text you typed in the field.

When processing the input we could parse the QUERY_STRING or the REQUEST_URI but Plack provides us a nicer way to do this. Plack provides a module called Plack::Request that provides a method called `param` which will return the value of a parameter sent by the user.

In order to simplify the code I created a function called `get_html` that returns a piece of static HTML. The form that will be displayed. The main code checks if the user has passed any parameter. If yes, the value is attached to the HTML we already have. This is what we have in the `$html` variable that we send back to the browser.

```
#!/usr/bin/perl
use strict;
use warnings;

use Plack::Request;

my $app = sub {
    my $env = shift;

    my $html = get_html();

    my $request = Plack::Request->new($env);

    if ($request->param('field')) {
        $html .= 'You said: ' . $request->param('field');
    }

    return [
```

```
            '200',
20.         [ 'Content-Type' => 'text/html' ],
21.         [ $html ],
        ];
    };

25. sub get_html {
        return q{
            <form>

            <input name="field">
30.         <input type="submit" value="Echo">
31.         </form>
            <hr>
        }
    }
```

Obviously for anything bigger we would move the HTML to a template file and we would probably even use a higher level Web framework such as Dancer or Mojolicious, but we are interested in the low level mechanism now.

A calculator?

If you'd like to take this approach a little bit further you could take the above script and enhance it to get two numbers and add them together.

An even more complex example would allow the user to provide two numbers and one of the basic operators (+, -, *, /) and return the result.

What else?

What else would you like to know about Plack and PSGI?