Creating REST Applications with the Slim Micro-Framework

Vikram Vaswani

October 2012

slim

adj. **slim·mer, slim·mest**

1. Small in girth or thickness in proportion to height or length; slender.
2. Small in quantity or amount; meager: slim chances of success.

From www.thefreedictionary.com

slim

PHP micro framework that helps you quickly write simple yet powerful web applications and APIs.

From www.slimframework.com

Micro-Frameworks in ~~20~~ 18 Words

- Lightweight
- Short learning curve
- Flexible
- Single-purpose
- Good for
  - Small, static websites
  - API development and testing
  - Rapid prototyping

PHP Micro-Frameworks

- Silex
- Limonade
- GluePHP
- FlightPHP
- Fat-Free Framework
- MicroMVC
- Bullet
- Bento
- Yolo
- Fluphy
- Tonic
- Phlyty
- Phraw
- Photon
- Fitzgerald
- phpMF
- Swiftlet
- Tachyon
- Pupcake
- Relax
- Lime
- Shield
- Hydra
- Gum

Entrée

Introducing Slim

"Slim is a PHP micro framework that helps you quickly write simple yet powerful web applications and APIs."

From www.slimframework.com

Key Features

- Powerful router
  - Standard and custom HTTP methods
  - Route parameters with wildcards and conditions
  - Route middleware
- Template rendering with custom views
- Flash messages
- Secure cookies with AES-256 encryption

Key Features

- HTTP caching
- Logging with custom log writers
- Error handling and debugging
- Middleware architecture
- Simple configuration
- Extensive documentation
- Active, enthusiastic community
- Licensed under the MIT Public License

First Taste

```php
<?php
// load
require 'Slim/Slim.php';
\Slim\Slim::registerAutoloader();

// initialize
$app = new \Slim\Slim();

// map routes
$app->get('/eat/:food', function ($food) {
  echo "Yum! That $food looks fantastic!";
});

// ...and we're off!
$app->run();
```

First Taste

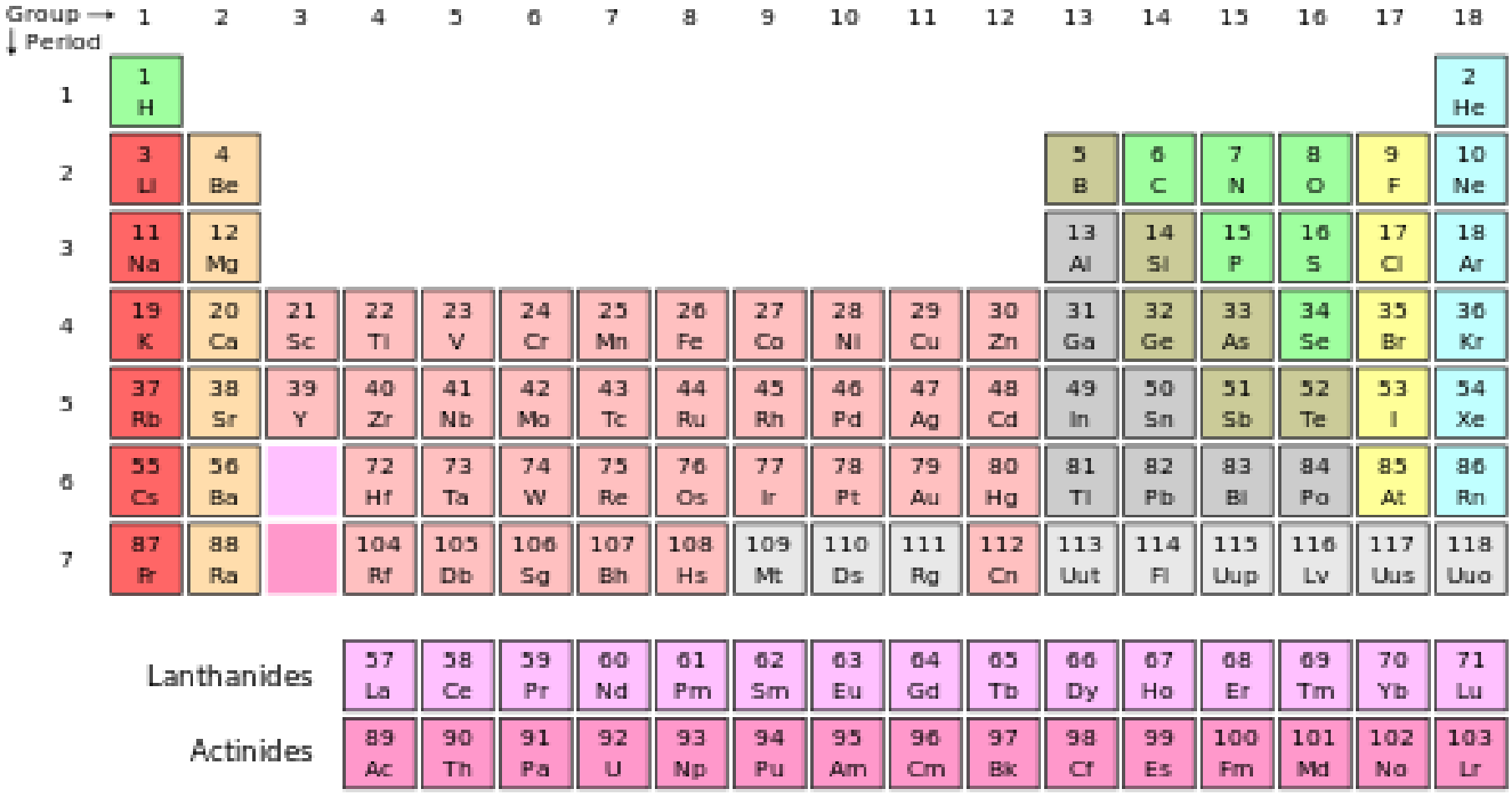

URL
Status
http://example.localhost/eat/ham
Headers
Response
Cache
HTML
Yum! That ham looks fantastic!

Main Course

Slim + REST

- Define URI routes, parameters and HTTP methods
  - GET / POST / PUT / DELETE
  - /my/api/endpoint/:parameter
- Map routes to callbacks
  - Accept and validate input parameters from request object
  - Perform custom processing within callback
  - Produce and return response object

Example: REST Resources



Source: Wikipedia
en.wikipedia.org/wiki/Periodic_table

REST API

GET `/api/v1/elements`	Retrieve all elements from the periodic table
GET `/api/v1/elements/1`	Retrieve element #1 (hydrogen)
POST `/api/v1/elements`	Create new element
PUT `/api/v1/elements/28`	Update element #28 (nickel)
DELETE `/api/v1/elements/8`	Delete element #8 (oxygen)

Resource Representation (MySQL)

```
CREATE TABLE IF NOT EXISTS `elements` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `num` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `symbol` char(2) NOT NULL,
  `weight` float NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

INSERT INTO `elements` (`id`, `num`, `name`, `symbol`,
`weight`) VALUES(1, 1, 'Hydrogen', 'H', 1.008);
INSERT INTO `elements` (`id`, `num`, `name`, `symbol`,
`weight`) VALUES(2, 2, 'Helium', 'He', 4.0026);
```

Resource Representation (JSON)


```
{
"id":"1",
"num":"1",
"name":"Hydrogen",
"symbol":"H",
"weight":"1.008"
},{
"id":"2",
"num":"2",
"name":"Helium",
"symbol":"He",
"weight":"4.0026"
}
```

RedBeanPHP

```php
<?php
// load
require 'RedBean/rb.php';


// set up database connection
R::setup('mysql:host=localhost;dbname=appdata','user',
'pass');
R::freeze(true);


// get all records
$elements = R::find('elements');
```

RedBeanPHP

```php
<?php

// get a single record

$element = R::findOne('elements', 'id=?',
array($id));


// update element record

$element->name = 'foo'


// save modified record

R::store($element);
```

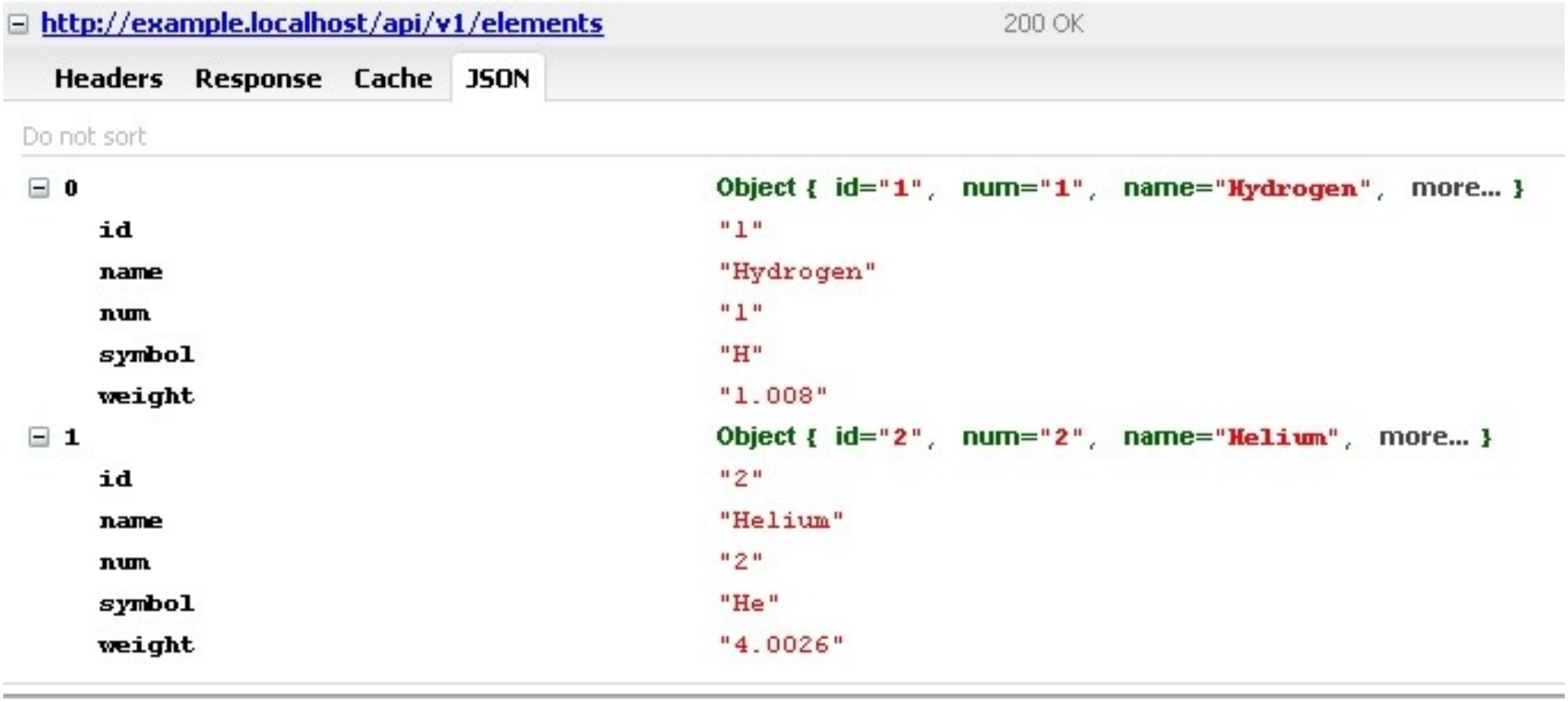
REST API Implementation

```php
<?php
$app->get('/api/v1/elements', function () { ... }

$app->get('/api/v1/elements/:id', function ($id) { ... }

$app->post('/api/v1/elements', function ($id) { ... }

$app->put('/api/v1/elements/:id', function ($id) { ... }

$app->delete('/api/v1/elements/:id', function ($id)
{ ... }
```

GET Request Handler

```php
<?php
// handle GET requests for /api/v1/elements
$app->get('/api/v1/elements', function () use ($app) {
    // get all elements
    $elements = R::find('elements');

    // create JSON response
    $app->response()->header('Content-Type',
'application/json');
    echo json_encode(R::exportAll($elements));
});
```

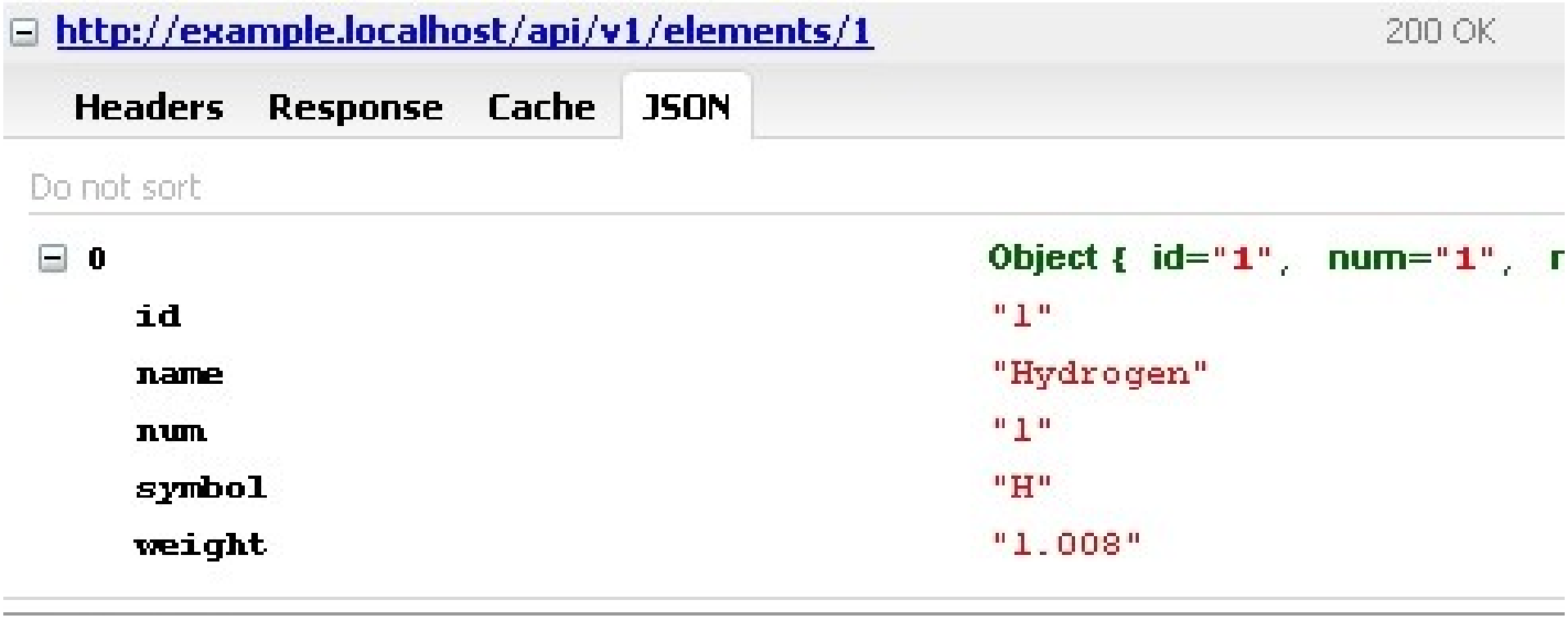
GET Response



GET Request Handler

```php
<?php
// handle GET requests for /api/v1/elements/*
$app->get('/api/v1/elements/:id', function ($id) use ($app) {
    // get element by id
    $article = R::findOne('elements', 'id=?', array($id));

    // create JSON response if element found
    // else send 404 server error
    if ($article) {
      $app->response()->header('Content-Type', 'application/json');
      echo json_encode(R::exportAll($article));
    } else {
      $app->response()->status(404);
    }
});
```

GET Response



In The News Recently…

Article:

2 New Elements on Periodic Table Get Names

Clara Moskowitz, LiveScience Senior Writer

Date: 01 June 2012 Time: 02:02 PM ET

Recommend 343 Tweet 45 +1 19 Share 6

Years after their discovery, the super-heavy elements 114 and 116 have finally been christened by their Russian and American discoverers. The elements

Two of the heaviest elements on the periodic table were officially named on Thursday (May 31).

The man-made elements 114 and 116, which contain 114 and 116 protons per atom, respectively, are now officially called flerovium (Fl) and livermorium (Lv).

The names were chosen to honor the laboratories that first created the elements: the

Source: LiveScience

www.livescience.com/20698-elements-periodic-table-flerovium-livermorium.html

POST Request Handler

```php
<?php
// handle POST requests for /api/v1/elements/*
$app->post('/api/v1/elements', function () use ($app) {
    // get request body, decode into PHP object
    $request = $app->request();
    $body = $request->getBody();
    $input = json_decode($body);

    // create and save element record
    $element = R::dispense('elements');
    $element->num = (string)$input->num;  // do same for other attributes
    R::store($element);

    // create and send JSON response
    $app->response()->status(201);
    $app->response()->header('Content-Type', 'application/json');
    echo json_encode(R::exportAll($element));
});
```

POST Request

http://example.localhost/api/v1/elements 201 Created

Headers | Post | Response | JSON

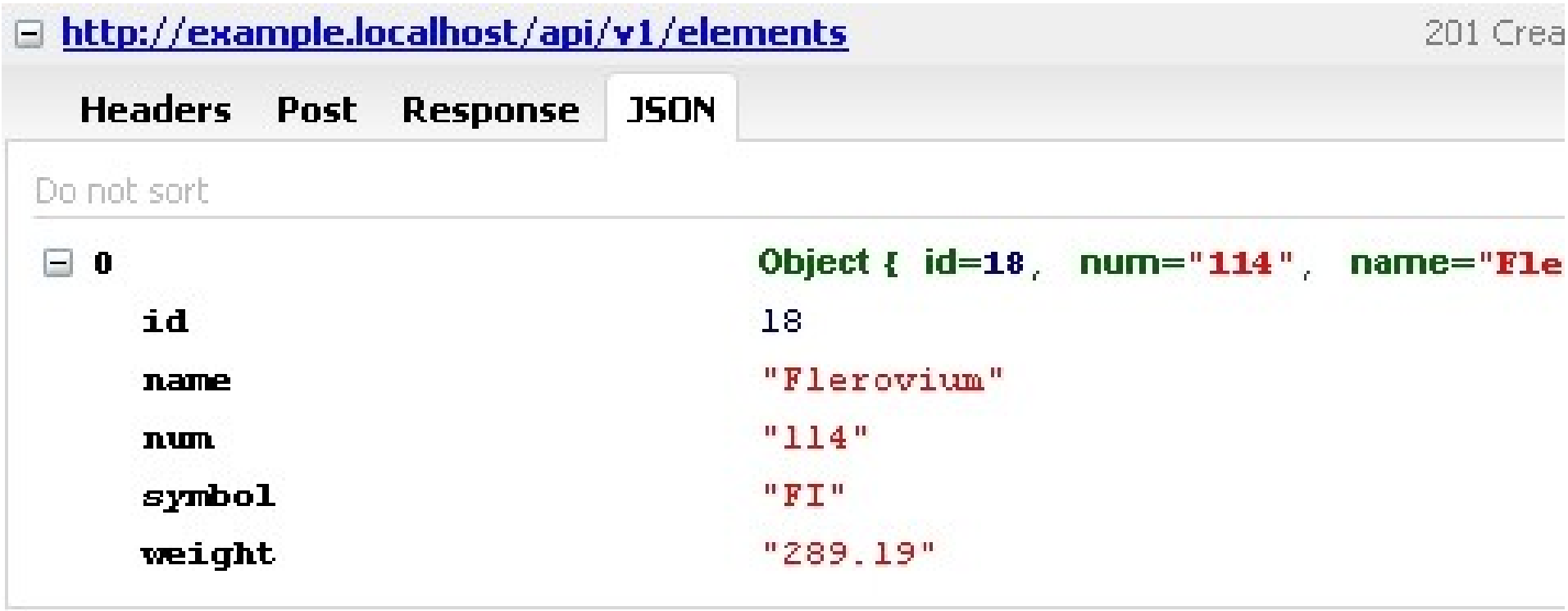
JSON

name	"Flerovium"
num	"114"
symbol	"Fl"
weight	"289.19"

Source

{"num":"114","name":"Flerovium","symbol":"Fl","weight":"289.19"}

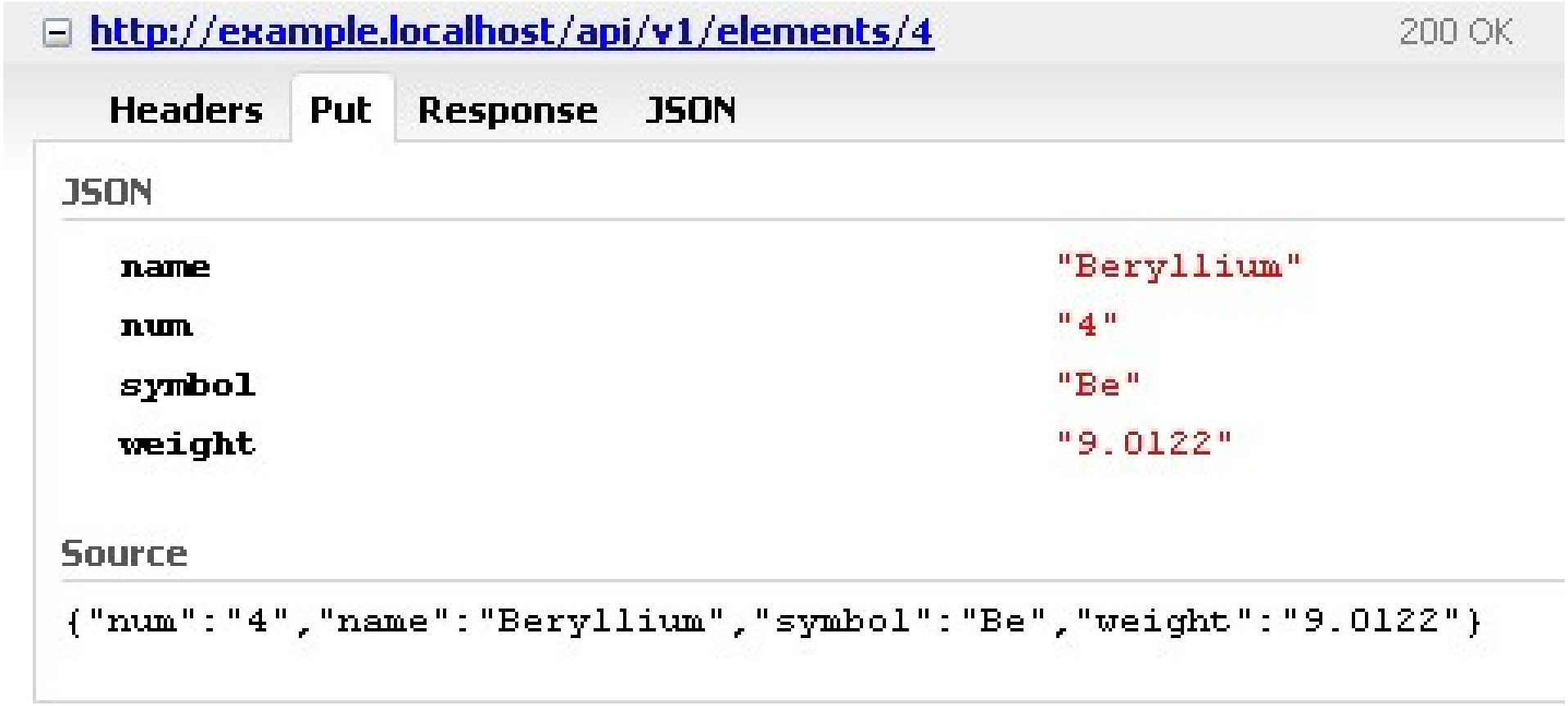
POST Response



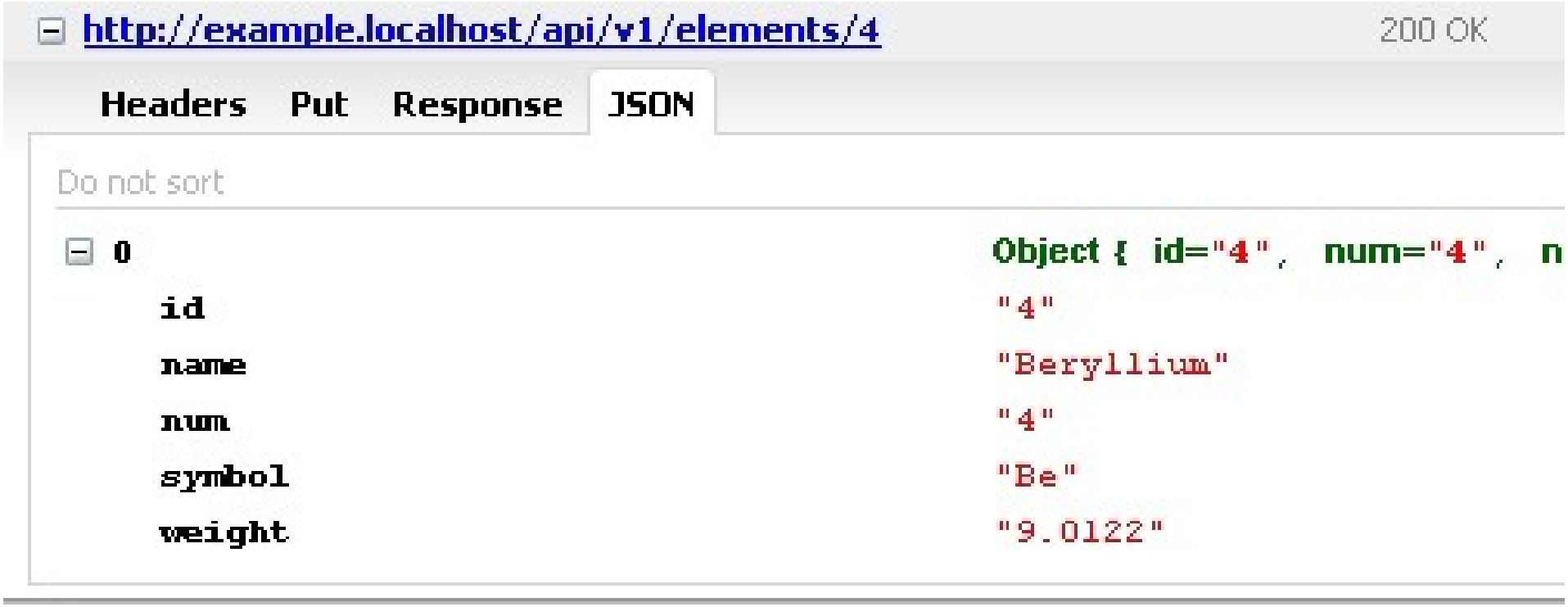
PUT Request Handler

```php
<?php
// handle PUT requests for /api/v1/elements/*
$app->put('/api/v1/elements/:id', function ($id) use ($app) {
    // get request body, decode into PHP object
    $request = $app->request();
    $body = $request->getBody();
    $input = json_decode($body);

    // retrieve specified element record
    // save modified record, create and send JSON response
    $element = R::findOne('elements', 'id=?', array($id));
    if ($element) {
      $element->num = (string)$input->num;  // do same for other attributes
      R::store($element);
      $app->response()->header('Content-Type', 'application/json');
      echo json_encode(R::exportAll($element));
    } else {
      $app->response()->status(404);
    }
});
```

PUT Request


http://example.localhost/api/v1/elements/4
200 OK
Headers
Put
Response
JSON
JSON
name
"Beryllium"
num
"4"
symbol
"Be"
weight
"9.0122"
Source
{"num":"4","name":"Beryllium","symbol":"Be","weight":"9.0122"}

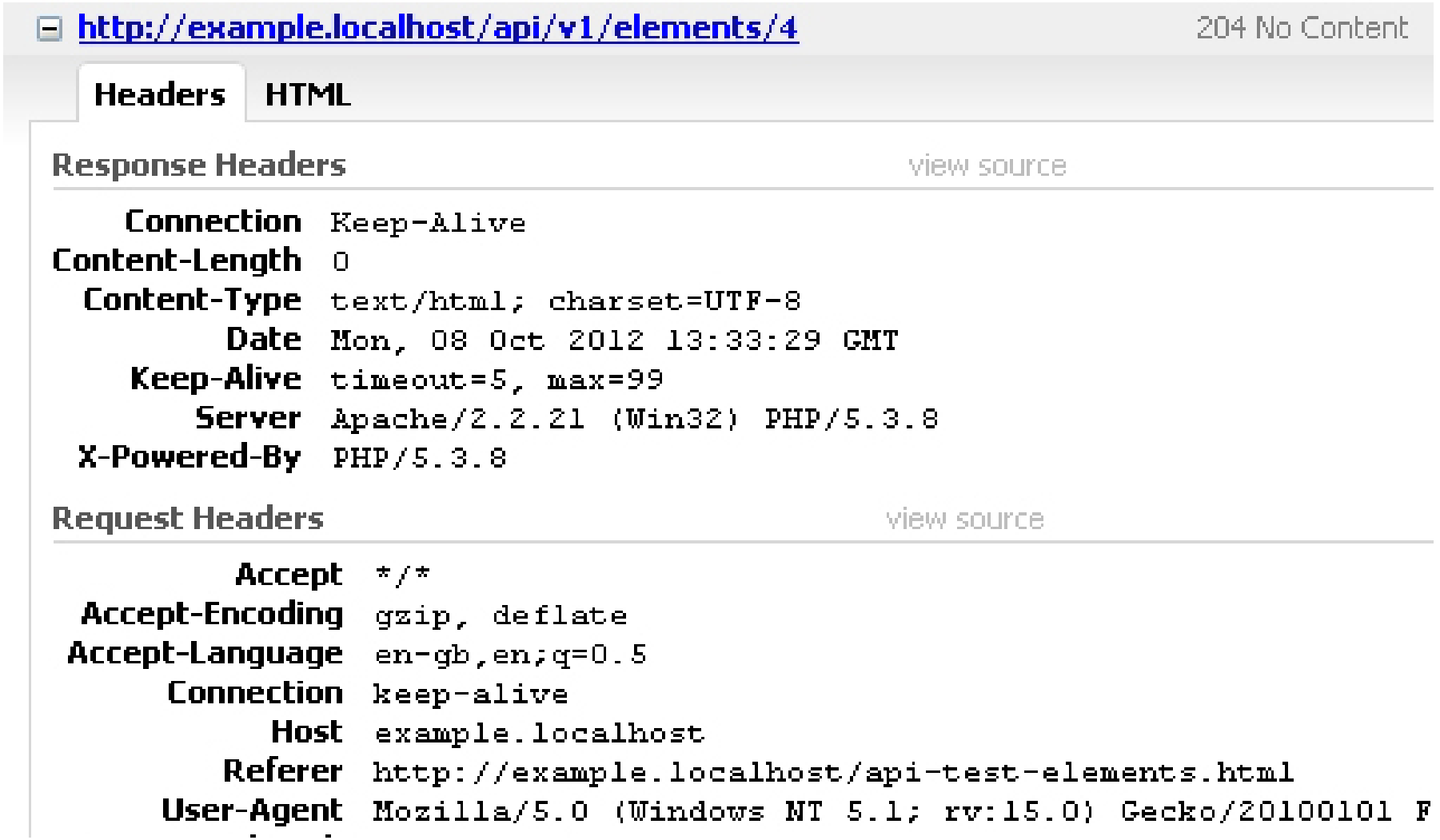
PUT Response



DELETE Request Handler

```php
<?php
// handle DELETE requests for /api/v1/elements
$app->delete('/api/v1/elements/:id', function ($id) use ($app) {
    $request = $app->request();
    $element = R::findOne('elements', 'id=?', array($id));
    if ($element) {
      R::trash($element);
      $app->response()->status(204);
    } else {
      $app->response()->status(404);
    }
});
```

DELETE Response



Accompaniments

Common Requirements

- Authentication
- Parameter validation
- Request logging
- Multi-format support

Common Requirements

- Authentication
- Parameter validation
- Request logging
- Multi-format support

Middleware

"The purpose of middleware is to inspect, analyze, or modify the application environment, request, and response before and/or after the Slim application is invoked."

From docs.slimframework.com

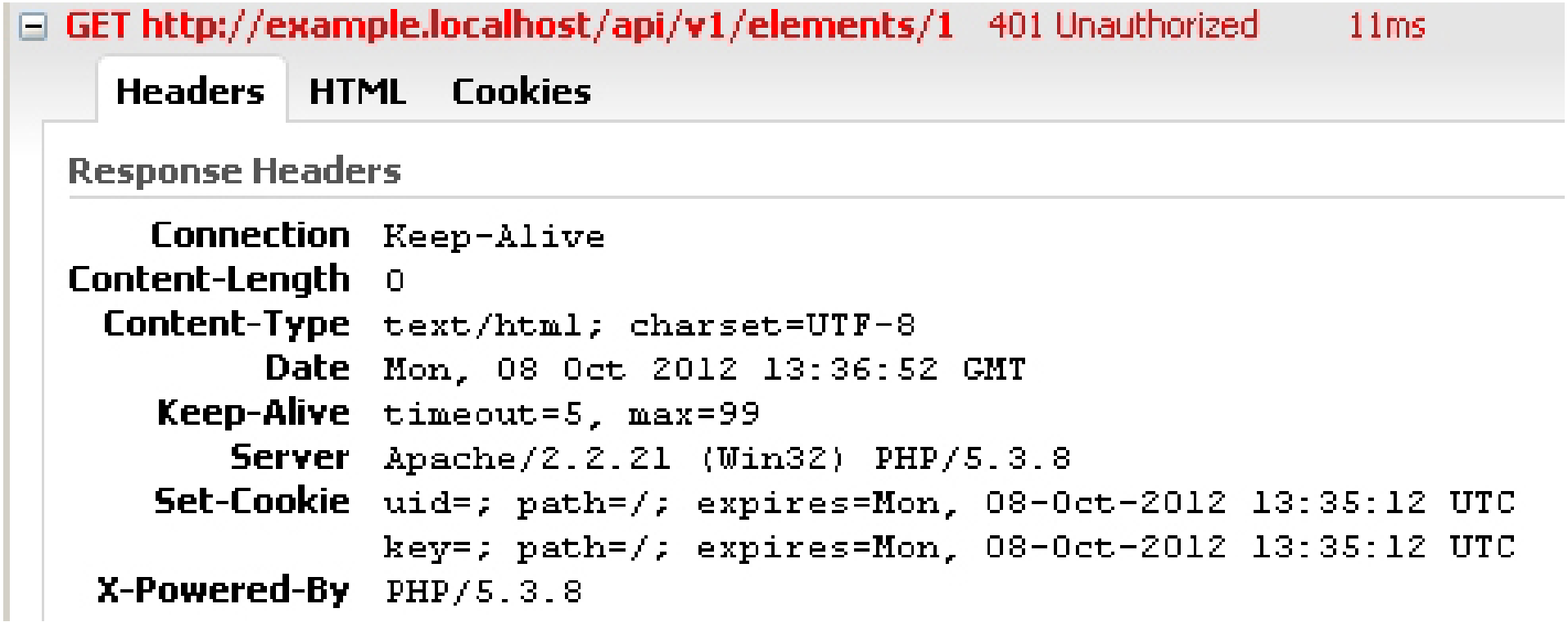
Middleware Use Cases

- Authentication/authorization
- Request pre-processing
- Response post-processing
- Filter chaining
- Logging

Example: API Authentication

```php
<?php
// route middleware for simple API authentication
function authenticate(\Slim\Route $route) {
    $app = \Slim\Slim::getInstance();
    $uid = $app->getEncryptedCookie('uid');
    $key = $app->getEncryptedCookie('key');
    if (validateUserKey($uid, $key) === false) {
      $app->halt(401);
    }
}

// add API authentication for GET requests
$app->get('/api/v1/elements', 'authenticate', function () use ($app) {
  // GET handler code here
});
```

Response to Unauthenticated API Request



Common Requirements

- Authentication
- Parameter validation
- Request logging
- Multi-format support

Example: API Parameter Validation

- Optional and mandatory route parameters:

```
<?php

$app->get('/api/v1/elements(/:symbol)', function
() use ($app) { ... };
```

- Route conditions:

```
<?php

$app->get('/api/v1/elements/:id', function () use
($app) { ... }->conditions(array('id' => '[0-9]
{1,}'));
```

Common Requirements

- Authentication
- Parameter validation
- Request logging
- Multi-format support

Request Object

```php
<?php
// get request object
$request = $app->request();

// get request headers as associative array
$headers = $request->headers();

// get request path
$type = $request->getPathInfo()

// get request IP address and host
$body = $request->getIp() . ',' . $request->getHost();
```

Example: API Request Logging

```php
<?php
// initialize logger
$app = new \Slim\Slim(array(
  'log.enabled' => 'true',
  'log.writer' => new FileLogWriter()
));

class FileLogWriter {
  public function write($message) {
    file_put_contents('my.log', $message . PHP_EOL,
FILE_APPEND);
  }
}
```

Example: Request Logging

```php
<?php
// handle GET requests for /api/v1/elements
$app->get('/api/v1/elements', function () use ($app) {
    // get request headers and log message
    $request = $app->request();
    $message = sprintf("%s [%s] \"%s %s\"",
      $request->getIp(),
      time(),
      $request->getMethod(),
      $request->getPathInfo()
    );
    $app->getLog()->info($message);

    // handler code
}
```

Common Requirements

- Authentication
- Parameter validation
- Request logging
- Multi-format support

Request Object

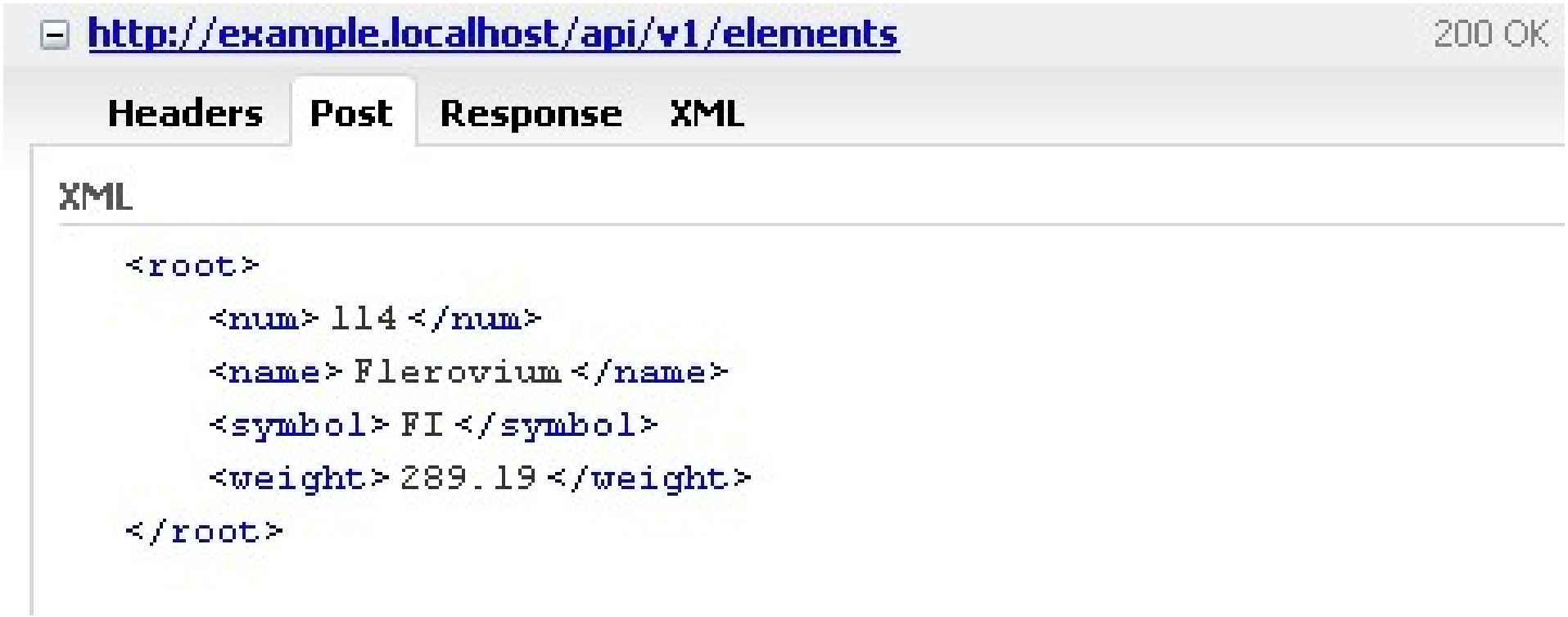
```php
<?php
// get request object
$request = $app->request();

// get request parameters as associative array
$headers = $request->params();

// get cookies
$headers = $request->cookies();

// get request content type
$type = $request->getMediaType()

// get raw request body
$body = $request->getBody();
```

Example: API Multi-Format Support

```php
<?php
$app->get('/api/v1/elements', function () use ($app) {
    // check request content type, send XML or JSON response
    $mediaType = $app->request()->getMediaType();
    if ($mediaType == 'application/xml') {
      $app->response()->header('Content-Type', 'application/xml');
      $xml = new SimpleXMLElement('<root/>');
      foreach (R::exportAll($elements) as $r) {
        $item = $xml->addChild('item');
        $item->addChild('id', $r['id']);  // do same for other attributes
      }
      echo $xml->asXml();
    } else if (($mediaType == 'application/json')) {
      $app->response()->header('Content-Type', 'application/json');
      echo json_encode(R::exportAll($elements));
    }
});
```

GET Response (XML)

POST Request (XML)



POST Response (XML)

http://example.localhost/api/v1/elements 200 OK

Headers Post Response XML

```
<root>
    <item>
        <id>8</id>
        <num>114</num>
        <name>Flerovium</name>
        <symbol>Fl</symbol>
        <weight>289.19</weight>
    </item>
</root>
```

Dessert

Performance

Put simply:

$F = ma$

Or, To Put It Another Way...

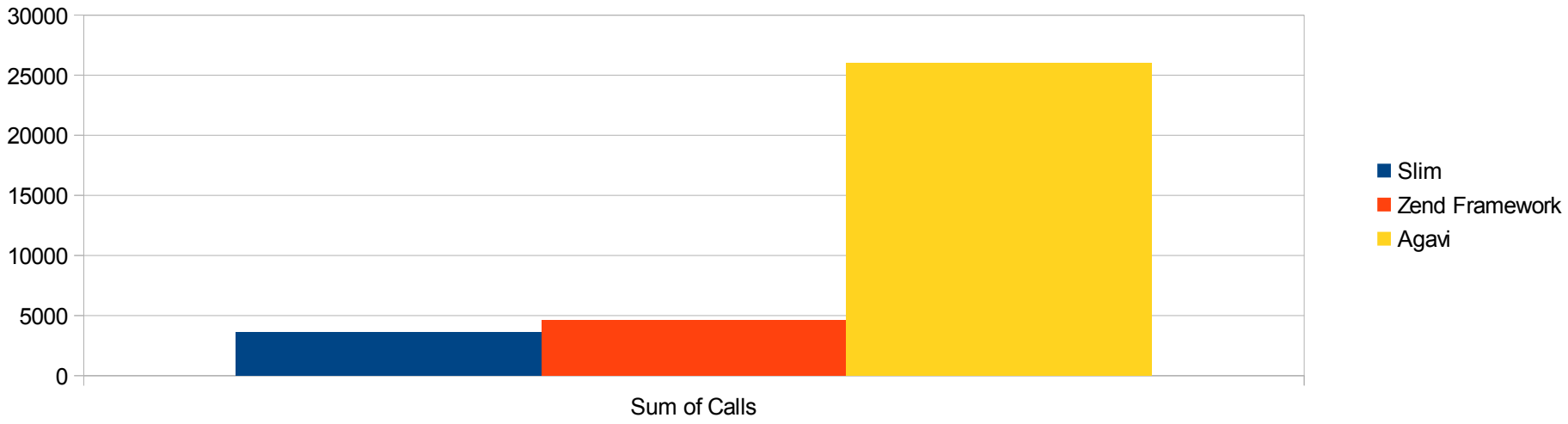
"If everything seems under control, you're not going fast enough."

- Mario Andretti

Source: Goodreads.com
http://www.goodreads.com/author/quotes/2115694.Mario_Andretti

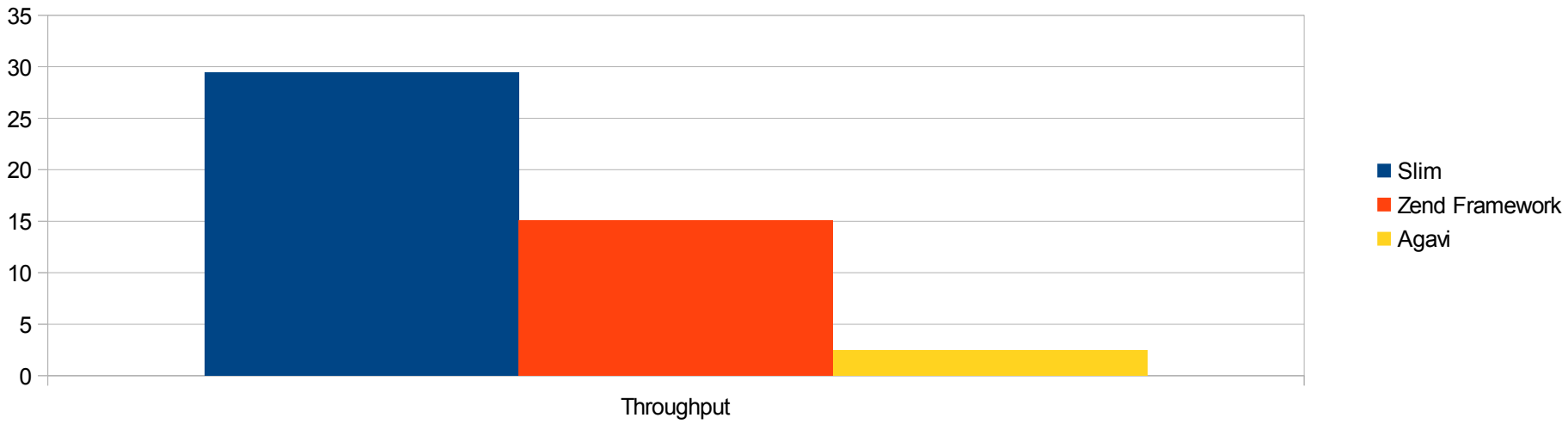
Weight

Framework	Self time (Milliseconds)	Sum of calls
Slim 2.0.0	1,008	3,612
Zend Framework 1.11	2,917	4,616
Agavi 1.0.1	13,885	25,988



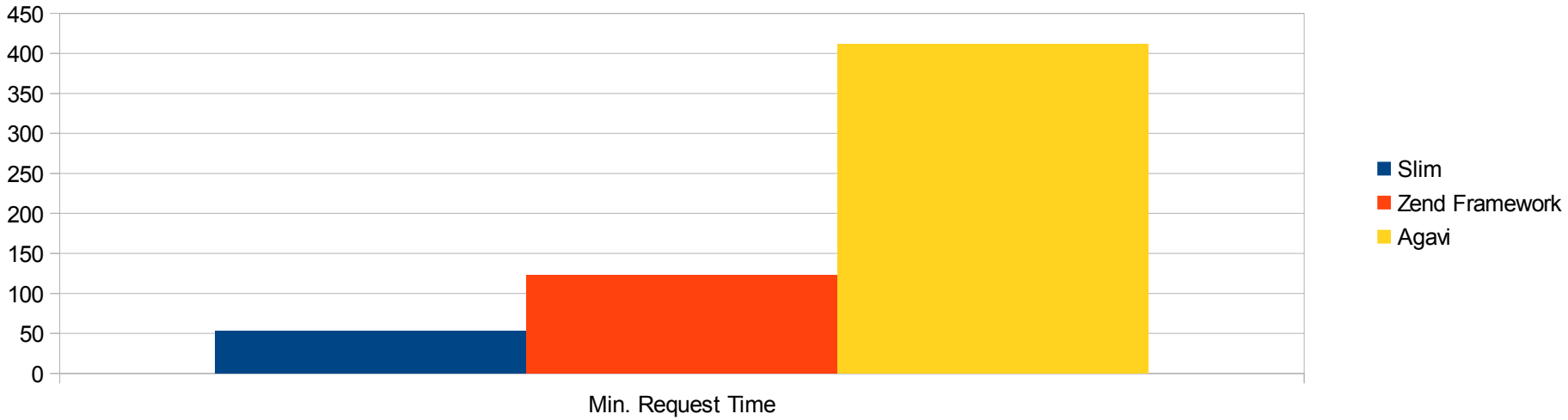
Requests Per Second

Framework	Throughput (Requests/Second)
Slim 2.0.0	29.43
Zend Framework 1.11	15.10
Agavi 1.0.1	2.44



Minimum Request Time

Framework	Min. Request Time (Milliseconds)
Slim 2.0.0	53
Zend Framework 1.11	123
Agavi 1.0.1	412



"Lies, Damned Lies and Statistics"

Test data based on informal testing

- Apache JMeter
  - 300 concurrent requests, averaged over 3 test runs
- Dual-core Dell server @ 2.4 GhZ
- Apache 2.2, PHP 5.3, Xdebug 2.2 and MySQL 5.1
- Unoptimized applications; default framework settings
- Your mileage may (and probably will) vary!

Resources

- Usage docs: docs.slimframework.com
- API docs: dev.slimframework.com/phpdocs
- Source code: github.com/codeguy/Slim
- Forums: help.slimframework.com
- News: www.slimframework.com/news
- Twitter: @slimphp

Questions?

Contact Information

Email:

- vikram-vaswani.in/contact

Web:

- www.melonfire.com
- vikram-vaswani.in

Social networks:

- plus.google.com/100028886433648406825