Writing APIs in Lumen

Rob Allen

June 2018

I write APIs

Let's start with Lumen

Lumen

- Microframework
- Built with Laravel components
- Fast!

Installation

Install Laravel Installer:

```
$ composer global require laravel/lumen-installer
```

Create a new project:

```
$ lumen new my-project-name
```

Create *.env* file

```
$ cd my-project-name
$ cp .env.example .env
```

Run using local webserver

```
$ php -S 127.0.0.1:8888 -t public/

$ curl http://localhost:8888
Lumen (5.6.3) (Laravel Components 5.6.*)
```

Differences from Laravel: Artisan

Artisan is pared down:

- No `serve`
- No `key:generate`
- No `tinker`
- No `env`
- No `down`
- ONLY `make:migration`
- No `make:model`
- No `make:controller`
- No `make:auth`
- etc

Add more Artisan!

1. Install flipbox's Lumen Generator:

```
$ composer require --dev flipbox/lumen-generator
```

2. Inside `bootstrap/app.php` file, add:

```
$app->register(Flipbox\LumenGenerator\
  LumenGeneratorServiceProvider::class);
```

Lumen-Generator

Provides these additional artisan commands:

```
make:command
make:event
make:listener
make:middleware
make:model
make:provider
make:test
make:controller
make:job
make:mail
make:migration
make:policy
make:seeder
key:generate
clear-compiled
serve
tinker
optimize
route:list
```

Differences from Laravel: Config

- No `config/` directory
- Configuration only via `.env`

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=
APP_TIMEZONE=UTC

LOG_CHANNEL=stack
LOG_SLACK_WEBHOOK_URL=
...
```

Differences from Laravel: app.php

- Register service providers in `bootstrap/app.php`

```
$app->register(App\Providers\AppServiceProvider::class);
```

- Register middleware in `bootstrap/app.php`

```
$app->middleware([
   App\Http\Middleware\ExampleMiddleware::class
]);

$app->routeMiddleware([
    'auth' => App\Http\Middleware\Authenticate::class,
]);
```

Differences from Laravel: Routing

Laravel (Symfony Router):

```
Route::get("/books/{id?}", function($id = null) {
    // do stuff
})->where('id', '[0-9]+');
```

Lumen (FastRoute):

```
$router->get('/books[/{id:[0-9]+}]',
  function ($id = null) {
    // do stuff
});
```

Disabled features

bootstrap/app.php:

```
// $app->withFacades();

// $app->withEloquent();
```

/ping: An API's "Hello World"

routes/web.php:

```
$router->get('/ping', function () {
    return response()->json(['ack' => time()]);
});
```

/ping: An API's "Hello World"

routes/web.php:

```
$router->get('/ping', function () {
    return response()->json(['ack' => time()]);
});
```

```
$ curl -i http://localhost:8888/ping
HTTP/1.0 200 OK
Host: localhost:8888
Content-Type: application/json

{"ack":1527326698}
```

What is an API?

What is a web API?

A server-side web API is a programmatic interface consisting of one or more publicly exposed endpoints to a defined request–response message system, typically expressed in JSON or XML, which is exposed via the web

Wikipedia

What is a web API?

- Programmatic interface
- Endpoints
- Request-response message system
- JSON or XML
- Stateless

What is REST?

- An *architecture*
- Centres on the transfer of *representations* of *resources*
  - A *resource* is any concept that can be addressed
  - A *representation* is typically a document that captures the current or intended state of a resource
- A *client* makes requests of a *server* when it wants to transition to a new state

Strengths

- Loose coupling
- Leverages the power of HTTP
- Emphasis on readability
  - HTTP methods as verbs: GET, POST, PUT, DELETE, etc.
  - Resources as nouns: collections and entities

Constraints: Client/Server

- Clients are not concerned with storage (portable)
- Servers are not concerned with UI or user state (scalable)

Constraints: Stateless

- No client context stored between requests. (no sessions!)

Constraints: Cacheable

- Safe idempotent methods are always cacheable
- Non-idempotent methods should allow clients to cache responses.
- Clients should honour HTTP headers with respect to caching.

Constraints: Layered system

- Client should not care whether it is connected directly to the server, or to an intermediary proxy.

Constraints: Uniform Interface

- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

Primary aspects of a RESTful API

- URI for *each* resource: `https://example.com/authors/1`
- HTTP methods are the set of operations allowed for the resource
- Media type used for representations of the resource
- The API must be hypertext driven

HTTP methods

Method	Used for	Idempotent?
GET	Retrieve data	Yes
PUT	Change data	Yes
DELETE	Delete data	Yes
POST	Change data	No
PATCH	Update data	No

Send `405 Method Not Allowed` not available for that resource

HTTP method negotiation

Lumen provides this for free!

```
$ curl -I -X DELETE http://localhost:8888

HTTP/1.0 405 Method Not Allowed
Host: localhost:8888
Allow: GET, POST
Connection: close
Content-type: text/html; charset=UTF-8
```

Routing in Laravel

routes/web.php:

```
$router->get('/authors/{id:\d+}', [
    'as' => 'author.list',
    'uses' => 'AuthorController@show'
]);
```

Routes have a method

Single methods:

```
$router->get()
$router->post()
$router->put()
$router->patch()
$router->delete()
$router->options()
```

Multiple methods:

```
$router->addRoute(['GET', 'POST'], …)
```

Routes have a pattern

- Literal string match

```
$router->get('/authors', …);
```

Routes have a pattern

- Literal string match

```
$router->get('/authors', …);
```

- Placeholders are wrapped in { }

```
$router->get('/authors/{id}', …);
```

Routes have a pattern

- Literal string match
```
$router->get('/authors', …);
```
- Placeholders are wrapped in { }
```
$router->get('/authors/{id}', …);
```
- Optional segments are wrapped with [ ]
```
$router->get('/authors[/{id}[/{books}]]', …);
```

Routes have a pattern

- Literal string match
```
$router->get('/authors', …);
```
- Placeholders are wrapped in { }
```
$router->get('/authors/{id}', …);
```
- Optional segments are wrapped with [ ]
```
$router->get('/authors[/{id}[/{books}]]', …);
```
- Constrain placeholders via Regex
```
$router->get('/authors/{id:\d+}', …); // digits
```

Routes have a name

Use `as` key to specify a name:

```
$router->get('/authors/{id:\d+}', [
    'as' => 'author.list',
    'uses' => 'AuthorController@show'
]);
```

Generate URL to named route:

```
$url = route('authors', ['id' => 1]);

// generates: /authors/1
```

Routes have an action

Use `uses` key to specify a controller:

```
$router->get('/authors/{id:\d+}', [
    'as' => 'author.list',
    'uses' => 'AuthorController@show'
]);
```

Action method in a controller

```php
namespace App\Http\Controllers;

use …;

class AuthorController extends Controller
{
    public function show(int $id)
    {
        $author = Author::findOrFail($id);
        return $author;
    }
}
```

Status codes

Send the right one for the right situation!

1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client error
5xx	Server error

Status codes are set in the Response

```
// AuthorController
public function add(Request $request): Response
{
    $data = $this->validate($request, [
        'name' => 'required|max:100',
    ]);

    $author = Author::create($data)->save();
    return response()->json($author, 201);
}
```

Status codes

```
$ curl -i -d name="Octavia Butler" \
    http://localhost:8888/authors

HTTP/1.0 201 Created
Host: localhost:8888
Content-Type: application/json

{"name":"Octavia Butler",
"updated_at":"2018-05-26 14:55:27",
"created_at":"2018-05-26 14:55:27",
"id":7}
```

Content negotiation

Correctly parse the request

- Read the `Content-Type` header
- Raise "415 Unsupported media type" status if unsupported

Correctly create the response

- Read the `Accept` header
- Set the `Content-Type` header

Handling unsupported types

```
class UnsupportedMiddleware
{
  public function handle($request, Closure $next)
  {
    $type = $request->headers->get('content-type');
    if (stripos($type, 'application/json') !== 0) {
        return response('Unsupported Media Type', 415);
    }
    return $next($request);
  }
}
```

Handling unsupported types

```
class UnsupportedMiddleware
{
  public function handle($request, Closure $next)
  {
    $type = $request->headers->get('content-type');
    if (stripos($type, 'application/json') !== 0) {
        return response('Unsupported Media Type', 415);
    }
    return $next($request);
  }
}
```

Handling unsupported types

```
class UnsupportedMiddleware
{
  public function handle($request, Closure $next)
  {
    $type = $request->headers->get('content-type');
    if (stripos($type, 'application/json') !== 0) {
        return response('Unsupported Media Type', 415);
    }
    return $next($request);
  }
}
```

Handling invalid Accept header

```
class UnacceptableMiddleware
{
  public function handle($request, Closure $next)
  {
    $accept = $request->headers->get('accept');
    if ($accept && stripos($accept, 'json') === false) {
      return response->json(['error' =>
        'You must accept JSON'], 406);
    }
    return $next($request);
  }
}
```

Handling invalid Accept header

```
class UnacceptableMiddleware
{
  public function handle($request, Closure $next)
  {
    $accept = $request->headers->get('accept');
    if ($accept && stripos($accept, 'json') === false) {
      return response->json(['error' =>
        'You must accept JSON'], 406);
    }
    return $next($request);
  }
}
```

Handling invalid Accept header

```
class UnacceptableMiddleware
{
  public function handle($request, Closure $next)
  {
    $accept = $request->headers->get('accept');
    if ($accept && stripos($accept, 'json') === false) {
      return response->json(['error' =>
        'You must accept JSON'], 406);
    }
    return $next($request);
  }
}
```

Handling invalid Accept header

```
$ curl -i -H "Accept: application/xml" http://localhost/
HTTP/1.0 406 Not Acceptable
Content-Type: application/json

{"error":"You must accept JSON"}
```

Hypermedia

- Media type used for a representation
- The link relations between representations and/or states
- Important for discoverability

JSON and Hypermedia

JSON does not have a defined way of providing hypermedia links

Options:

- "Link" header (GitHub approach)
- application/collection+json
- application/hal+json
- JSON-API

Fractal

- Separate the logic for your JSON formation from your Eloquent model
- Supports multiple serialisers including JSON-API
- Install:

```
$ composer require league/fractal
```

Fractal service provider

```
class FractalManagerProvider extends ServiceProvider
{
  public function register()
  {
    $this->app->singleton(Manager::class,function($app) {
      $manager = new Manager();
      $base = app(Request::class)->getBaseURL();
      $manager->setSerializer(new JsonApiSerializer($base));
      return $manager;
    });
  }
}
```

Fractal service provider

```
class FractalManagerProvider extends ServiceProvider
{
  public function register()
  {
    $this->app->singleton(Manager::class,function($app) {
      $manager = new Manager();
      $base = app(Request::class)->getBaseURL();
      $manager->setSerializer(new JsonApiSerializer($base));
      return $manager;
    });
  }
}
```

Fractal service provider

```
class FractalManagerProvider extends ServiceProvider
{
  public function register()
  {
    $this->app->singleton(Manager::class,function($app) {
      $manager = new Manager();
      $base = app(Request::class)->getBaseURL();
      $manager->setSerializer(new JsonApiSerializer($base));
      return $manager;
    });
  }
}
```

Fractal service provider

```
class FractalManagerProvider extends ServiceProvider
{
  public function register()
  {
    $this->app->singleton(Manager::class,function($app) {
      $manager = new Manager();
      $base = app(Request::class)->getBaseURL();
      $manager->setSerializer(new JsonApiSerializer($base));
      return $manager;
    });
  }
}
```

AuthorTransformer

```php
class AuthorTransformer extends Fractal\TransformerAbstract
{
    public function transform(Author $author)
    {
        return [
            'id' => (int) $author->id,
            'name' => $author->name,
        ];
    }
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Create JSON-API response

```
public function list(Manager $fractal) : Response
{
    $authors = Author::all();

    $resource = new Collection($authors,
        new AuthorTransformer, 'authors');
    return response()->json(
        $fractal->createData($resource)->toArray(),
        200,
        ['content-type' => 'application/vnd.api+json']
    );
}
```

Output

```
$ curl http://localhost:8888/authors/4
{
  "data": [
      {
          "type": "authors",
          "id": "1",
          "attributes": {
              "name": "Suzanne Collins",
          },
          "links": {
              "self": "/authors/1"
          }
      },
```

When things go wrong

Default error output

```
$ curl http://localhost:8888/authors/999
<!DOCTYPE html>
<html>
    <head>
        <meta name="robots" content="noindex,nofollow" />
        <style>
          /* Copyright (c) 2010, Yahoo! Inc. All rights reser
          html{color:#000;background:#FFF;}body,div,dl,dt,dd,
          html { background: #eee; padding: 10px }
          img { border: 0; }
          #sf-resetcontent { width:970px; margin:0 auto; }
```

Great error handling

- Error representations are first class citizens
- Provides application error code & human readable message
- Pretty prints for the humans!

Override ExceptionsHandler::render()

```php
public function render($request, Exception $e)
{
  $statusCode = $this->getStatusCodeFromException($e);
  $error['error'] = Response::$statusTexts[$statusCode];
  if (env('APP_DEBUG')) {
    $error['message'] = $e->getMessage();
    $error['file'] = $e->getFile() . ':' . $e->getLine();
    $error['trace'] = explode("\n", $e->getTraceAsString());
  }

  return response()->json($error, $statusCode, [],
    JSON_PRETTY_PRINT|JSON_UNESCAPED_SLASHES);
}
```

Override ExceptionsHandler::render()

```
public function render($request, Exception $e)
{
  $statusCode = $this->getStatusCodeFromException($e);
  $error['error'] = Response::$statusTexts[$statusCode];
  if (env('APP_DEBUG')) {
    $error['message'] = $e->getMessage();
    $error['file'] = $e->getFile() . ':' . $e->getLine();
    $error['trace'] = explode("\n", $e->getTraceAsString());
  }

  return response()->json($error, $statusCode, [],
    JSON_PRETTY_PRINT|JSON_UNESCAPED_SLASHES);
}
```

Override ExceptionsHandler::render()

```
public function render($request, Exception $e)
{
  $statusCode = $this->getStatusCodeFromException($e);
  $error['error'] = Response::$statusTexts[$statusCode];
  if (env('APP_DEBUG')) {
    $error['message'] = $e->getMessage();
    $error['file'] = $e->getFile() . ':' . $e->getLine();
    $error['trace'] = explode("\n", $e->getTraceAsString());
  }

  return response()->json($error, $statusCode, [],
    JSON_PRETTY_PRINT|JSON_UNESCAPED_SLASHES);
}
```

Override ExceptionsHandler::render()

```
public function render($request, Exception $e)
{
  $statusCode = $this->getStatusCodeFromException($e);
  $error['error'] = Response::$statusTexts[$statusCode];
  if (env('APP_DEBUG')) {
    $error['message'] = $e->getMessage();
    $error['file'] = $e->getFile() . ':' . $e->getLine();
    $error['trace'] = explode("\n", $e->getTraceAsString());
  }

  return response()->json($error, $statusCode, [],
    JSON_PRETTY_PRINT|JSON_UNESCAPED_SLASHES);
}
```

Error output (live)

```
$ curl -i http://localhost:8888/authors/999
HTTP/1.0 404 Not Found
Content-Type: application/json

{
  "error": "Not Found"
}
```

Error output (debug)

```
$ curl -i http://localhost:8888/authors/999
HTTP/1.0 404 Not Found
Content-Type: application/json

{
  "error": "Not Found",
  "message": "No query results for model [App\\Author] 999",
  "file": "vendor/illuminate/database/Eloquent/Builder.php:33
  "trace": [
      "#0 vendor/illuminate/database/Eloquent/Model.php(1509)
      "#1 vendor/illuminate/database/Eloquent/Model.php(1521)
      "#2 app/Http/Controllers/AuthorController.php(30): Illu
…
```

To sum up

Resources

- https://github.com/akrabat/lumen-bookshelf-api
- https://lumen.laravel.com/docs/
- https://fractal.thephpleague.com

Books:

- *Build APIs You Won't Hate* by Phil Sturgeon
- *RESTful Web APIs* by L. Richardson, M. Amundsen & S. Ruby

Thank you!