

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| **Document title** | **PHPSaaS WRAPPER** |
| **Document reference number** | **UEL -** |
| **Project name** | **UNISA e-Content 20140363** |
| **Unisa Department** | **Department of Economics** |

Table of Contents

1 Description of the component 5

2 How to Install The component 5

3 how to use the component 12

Document sign-off

This document has been seen and accepted by the following people:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Title | Date | Signature |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Compiler | Purpose | Change Summary |
| <yyyymmdd> | Name Surname | Document Created,  Internal Review,  External Review,  etc | Initial release,  Updates based on internal review,  Updates to section xxx,  etc |
|  |  |  |  |
|  |  |  |  |

# Description of component

This is a software as a service software written in php

# How to Install PHPSaasWrapper

Step 1 Installation:

Step 1.1

Install Laravel framework. The following command will install the latest version of Laravel with the project name LaravelDefault.

composer create-project laravel/laravel LaravelDefault

Step 1.2

Download Zip from [PHPSaasWrapper Git repo](https://github.com/EonConsulting/PHPSaasWrapper).

Step 2 Configuration:

Step 2.1

Create a folder called "packages" inside of the root of the Laravel project.

Step 2.2

Paste the folder from the Git repo (Step 1.2) inside the packages folder.

Step 2.3

Inside the root Laravel project, open the composer.json file. In that file, look for:

"autoload" : {

"psr-4": {

...

Inside of the psr-4 tag, load the following:

"EONConsulting\\PHPSaasWrapper\\": "packages/PHPSaasWrapper/src",

Step 2.4 Use as a global function instead of by namespace

Under the psr-4 but still inside of the autoload if the array files does not exist, create it like this:

"files": [

...

]

And enter into it:

"packages/PHPSaasWrapper/src/Http/helpers.php",

"packages/PHPSaasWrapper/vendor/guzzlehttp/guzzle/src/functions.php",

"packages/PHPSaasWrapper/vendor/guzzlehttp/psr7/src/functions.php",

"packages/PHPSaasWrapper/vendor/guzzlehttp/promises/src/functions.php",

Step 3 Registering the Service Provider

Step 3.1

Open up config/app.php. In the providers array, enter the following:

EONConsulting\PHPStencil\PHPStencilServiceProvider::class,

Step 3.2

In the same file, in the aliases array, enter:

'PHPSaasWrapper' => \EONConsulting\PHPSaasWrapper\src\Facades\PHPSaasWrapper::class,

Step 4 Tsugi

To be able to use the tsugi, the APP\_URL in your env file needs to be correct. So change it to the URL you are using for your testing environment.

Step 5 the Finale

In the command line / terminal, enter the following command in the root of the Laravel project:

composer dump-autoload -o

Migrations

We have included two tables in the database. These include ServiceAvailable and ServiceLinked. To have access to the migrations, you will need to perform the following commands:

php artisan vendor:publish

php artisan migration

Testing

Let's see this in action. Go to routes/web.php and where it says return view ('welcome'); replace that with

echo PHPSaasWrapper::output();

// or

echo phpsaaswrapper()->output();

To test this out, go to the URL for this Laravel site.

You should now see all of the api's being listed.

Config

The PHPSaasWrapper has been built to store the api information in the config.php in the EONConsulting\PHPSaasWrapper\src\Factories namespace.

All the API's you wish to harness will need to go into the $data array. The structure is as follows:

'oauth' => [

'return\_uri' => 'http://project\_url.dev/\_eon\_phpsaaswrapper/auth/callback',

'allows' => [

'api\_name\_1' => [

'requires' => [

'client\_id',

'redirect\_uri'

],

'client\_id' => '123abc',

'secret' => 'abc123',

'requires\_auth' => 'true',

'redirect\_uri' => 'https://example.com/login/oauth/authorize',

'redirect\_uri\_skeleton' => 'https://example.com/login/oauth/authorize?client\_id=--client\_id--&redirect\_uri=--redirect\_uri--',

'access\_token\_uri' => 'https://example.com/login/oauth/access\_token',

],

'api\_name\_2' => [

'requires' => [

'user\_key',

'output'

],

'user\_key' => '123',

'client\_id' => '123',

'secret' => '',

'requires\_auth' => 'false',

'redirect\_uri' => '',

'redirect\_uri\_skeleton' => '',

'access\_token\_uri' => '',

'output' => 'json',

'api\_uses' => [

'endpoint\_group\_1' => [

'endpoint\_1\_key',

'endpoint\_2\_key',

'endpoint\_3\_key',

],

'endpoint\_group\_2' => [

'endpoint\_4\_key',

'endpoint\_5\_key',

'endpoint\_6\_key'

],

'endpoint\_group\_3' => [

'endpoint\_7\_key'

]

],

'api\_links' => [

'endpoint\_1\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_2\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_3\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_4\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_5\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_6\_key' => [

'uri' => 'http://example.com/example?key=--user\_key--&recipe=--recipe--&output=--output--',

'requires' => [

'user\_key',

'recipe',

'output'

]

],

'endpoint\_7\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

],

'templates' => [

'endpoint\_1\_key' => '\_endpoint\_1\_key\_template\_name',

'endpoint\_2\_key' => '\_endpoint\_2\_key\_template\_name',

'endpoint\_3\_key' => '\_endpoint\_3\_key\_template\_name'

]

],

'api\_name\_3' => [

...

Now let's give a real example to explain what is going on here.

We are going to implement 2 API's with different ways of exposing endpoints. The first API we are going to consume is [Harvards' CS50 API](https://cs50.3scale.net/) and then the second is [UBC Library's Open Collection API](https://open.library.ubc.ca/docs).

The first API is pretty simple, simple endpoints, simple use. The second API is more complex, having multiple endpoints for each item. We will go through what I mean by this later.

Setup an API config

Let's have a look at how to setup an API config.

The general rule of thumb with the PHPSaaSWrapper is that it goes 'oauth.allows.<insert api name here>'. That api name is a key. It needs to be a slug. It is also just for reference, so it does not need to be official by the API themselves, this just is so we know when to use which API.

Generally I try to put the default needed data at the top of the array and the content that changes towards the bottom.

requires

'api\_name\_1' => [

'requires' => [

'client\_id',

'redirect\_uri'

],

The requires key that goes in the root of the api is a default / fallback for all the api links / uri's. If most URL's require the client\_id and redirect\_uri to be passed through, stick it in here to avoid putting a requires array on all of the endpoints. You can have any keys go in here, no problem.

Anything that is in the requires array, be it in the root or the actual endpoint, will need to be in the root of the api, ie:

'api\_name\_1' => [

'client\_id' => '123',

'redirect\_uri' => 'http://...'

user\_key && client\_id

'user\_key' => '123',

'client\_id' => '123',

Due to different API's calling they main key different names (API key, user key, client key, etc.) We have used client\_id for ours. But you can use any of those you like. Just make sure that you put in the key into the client\_id field.

**Authentication**

'secret' => '',

'requires\_auth' => 'false',

'redirect\_uri' => '',

'redirect\_uri\_skeleton' => '',

'access\_token\_uri' => '',

The PHPSaaSWrapper is build to use OAuth on API's that require authentication other than passing a key.

If the API requires authentication, set the requires\_auth to true, otherwise set it to false.

then we have a redirect\_uri, redirect\_uri\_skeleton and access\_token\_uri.

The redirect\_uri and redirect\_uri\_skeleton are the same endpoint, except the skeleton has the GET parameters that it will require.

It will make a request to the redirect\_uri and if it validates, then it will make a request to receive an access token from the access\_token\_uri.

### API Uses

Simple use

'api\_uses' => [

'endpoint\_group\_1' => [

'endpoint\_1\_key',

'endpoint\_2\_key',

'endpoint\_3\_key',

],

'endpoint\_group\_2' => [

'endpoint\_4\_key',

'endpoint\_5\_key',

'endpoint\_6\_key'

],

'endpoint\_group\_3' => [

'endpoint\_7\_key'

]

],

Complex use

'api\_uses' => [

'endpoint\_1\_key' => ['id' => 'endpoint\_id', 'label' => 'Title for label', 'use' => 'id'],

'endpoint\_2\_key' => ['id' => 'endpoint\_id', 'label' => 'Title for label', 'use' => 'id'],

],

We now come to specifying the uses of the API. These can be straight endpoints specified, or they can be wrapped in groups. This will just be the key for the actual link, it won't be the label and uri, etc.

Simple use explanation: You can list the endpoints simply by key that responds to the relevant api\_links key.

Complex use explanation: In the use parameter, you specify what the key is, and then you can specify the label to show up on the frontend in the label tag.

### API Links

Simple use

'api\_links' => [

'endpoint\_1\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_2\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_3\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_4\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_5\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

'endpoint\_6\_key' => [

'uri' => 'http://example.com/example?key=--user\_key--&recipe=--recipe--&output=--output--',

'requires' => [

'user\_key',

'recipe',

'output'

]

],

'endpoint\_7\_key' => 'http://example.com/example?key=--user\_key--&output=--output--',

],

Complex Use

'api\_links' => [

'endpoint\_1\_key' => [

'collection\_metadata' => [

'uri' => '--base\_url--/collections/46343',

'label' => 'Collection Metadata',

'requires' => [

'base\_url'

]

],

'items' => [

'uri' => '--base\_url--/collections/46343/items',

'label' => 'Items',

'requires' => [

'base\_url'

]

],

'items\_metadata' => [

'uri' => '--base\_url--/collections/46343/items/--item\_id--',

'label' => 'Item Metadata',

'requires' => [

'base\_url',

'item\_id'

]

],

'total' => [

'uri' => '--base\_url--/collections/46343/\_total',

'label' => 'Total Results Count',

'requires' => [

'base\_url'

]

],

],

'endpoint\_2\_key' => [

The api\_links is to specify the actual URI's for the endpoints. For the URI you can specify this by just passing it through as a string, or you can pass through an array with a custom requires key for just that URI.

When it generates the API links, it will 'build' these uri's to the full string. Remember, whatever is in the requires array must be available in the root of the api.

### Tsugi Config

We have provided a very easy way to use Tsugi.

You just need to setup the keys in your .env file. Like so:

TSUGI\_DB=tsugi

TSUGI\_DB\_HOST=127.0.0.1

TSUGI\_DB\_USER=root

TSUGI\_DB\_PASS=root

TSUGI\_DB\_PREFIX=""

TSUGI\_ADMIN\_PASS=1234

TSUGI\_SERVICE\_NAME="TSUGI"

TSUGI\_SERVICE\_DESC=""

TSUGI\_OWNER\_NAME="Charles Severance"

TSUGI\_OWNER\_MAIL=""

TSUGI\_PROVIDE\_KEYS=false

TSUGI\_GOOGLE\_CLIENT\_ID=false

TSUGI\_GOOGLE\_CLIENT\_SECRET=false

TSUGI\_GOOGLE\_MAP\_API\_KEY=false

TSUGI\_BADGE\_ENCRYPT\_PASSWORD=false

TUSIG\_BADGE\_ASSERT\_SALT=false

TSUGI\_PRODUCT\_INSTANCE\_GUID="lti2.example.com"

TSUGI\_DEVELOPER=true

TSUGI\_COOKIE\_SECRET="warning:please-change-cookie-secret-a289b543"

TSUGI\_COOKIE\_NAME="TSUGIAUTO"

TSUGI\_COOKIE\_PAD="390b246ea9"

TSUGI\_MAIL\_DOMAIN=false

TSUGI\_MAIL\_SECRET="warning:please-change-mailsecret-92ds29"

TSUGI\_MAILEOL="\n"

TSUGI\_NONCE\_CHECK=100

TSUGI\_NONCE\_TIME=1800

TSUGI\_SESSION\_SALT="warning:please-change-sessionsalt-89b543"

TSUGI\_TIMEZONE="Pacific/Honolulu"

TSUGI\_OLD\_ANALYTICS\_KEY=false

TSUGI\_OLD\_ANALYTICS\_NAME=false

TSUGI\_UNIVERSAL\_ANALYTICS=false

TSUGI\_OFFLINE=false

Fill out these config details to best suit your project. If you do not know what the key means or what to change it to, leave it to its default value above.

# How to use PHPSaasWrapper

This is pretty easy. There is no need to create your own controllers or logic. We have set up routes in our package to handle this for you.

In this example we are going to use the domain eon.dev to demonstrate the use.

Simply navigate to eon.dev/\_eon/<api-key-in-config-here> and you will see the uses for the requested api.

For example, to access the CS50 API, we just need to go to eon.dev/\_api/cs50 and it will show us all the possible endpoints we have access to.

**3.1 Templating API Responses**

Setting up the template

We have provided a handy method for providing templates to render on use of an API endpoint.

First, take note of the lowest endpoint name.

api\_uses' => [

'courses' => [

'courses',

'faculty',

'fields',

],

'facts' => [

'recipes',

'menus',

'facts'

],

'maps' => [

'buildings'

]

],

in this case the template keys would be:

'courses' => '',

'faculty' => '',

'fields' => '',

'recipes' => '',

'menus' => '',

'facts' => '',

'buildings' => '',

Inside of the API Key in Config.php, create a templates array if it does not already exist.

'cs50' => [

'templates' => []

Inside of the templates key, add the template keys from above:

'templates' => [

'courses' => '',

'faculty' => '',

'fields' => '',

'recipes' => '',

'menus' => '',

'facts' => '',

'buildings' => '',

]

Now create your template files inside of your resources/views/ folder. *NOTE* These templates need to be inside resources/views/templates/<api-key-from-config>, eg: resources/views/templates/cs50.

Then just make sure that whatever template you create, you use the exact same name (case sensitive) inside the templates array.

So if you created a template to render a course called \_course.blade.php, then you need add the filename \_course to the templates array.

'templates' => [

'courses' => '\_course',

## 3.2 Getting data inside of the template

The data passed from the api will be passed through in a variable called $response. So if your API passed through a name and email with each item, access them like this:

$name = $request->name;

$email = $request->email;

## Handy Functions

The PHPSaasWrapper comes with some handy functions, easy for extending the package for your own needs.

## Authentication

## authorize

If you have an API that you know requires authorization via OAuth, simply call the authorizemethod and pass through the key of the API.

return phpsaaswrapper()->authorize('github');

It will then do the authorization redirects and everything for you.

## needs\_auth

Not sure if an API needs authentication? Use the needs\_auth method to check.

return phpsaaswrapper()->needs\_auth('github');

## index

Need to get all of the API's listed? Easy. Just call the index() method.

return phpsaaswrapper()->index();

## display\_api\_uses

To display the uses of a specific API in HTML tree format, call the display\_api\_uses method.

return phpsaaswrapper()->display\_api\_uses('cs50');

# Landscape layout

Example of landscape layout

Appendix A

For Appendices use the style “UEL Appendix”