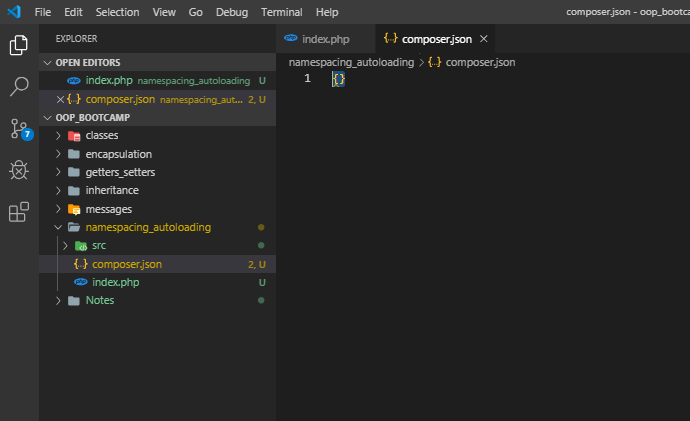
Composer, autoloading component and namespaces

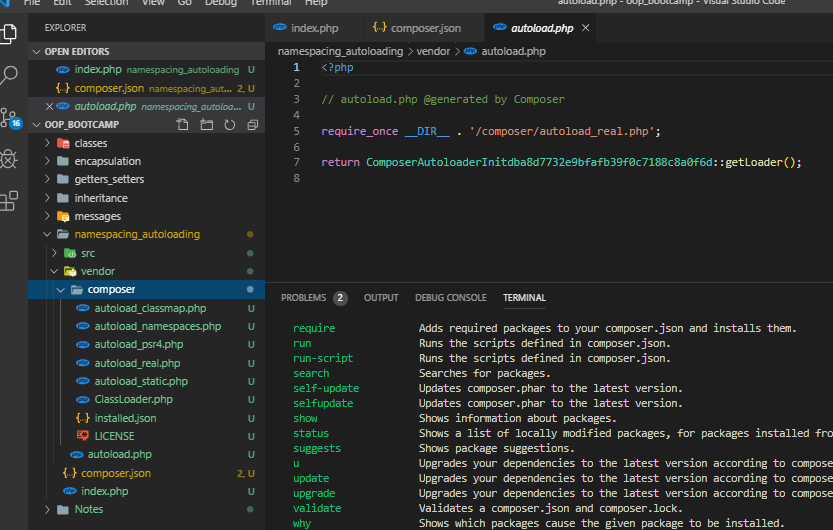
Composer is a package manager for php (similar to npm/yarn for JS).

It has an awesome autoloading component.

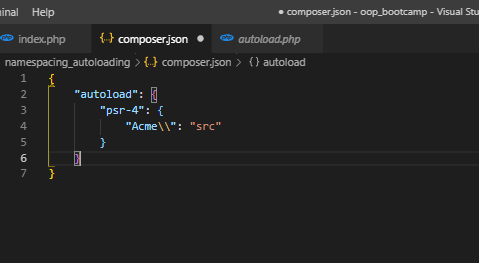
In order to set it up we need to create a ***composer.json*** file:



Next run ***composer install*** (this will set up the autoload component – autoload.php + other files required by the autoload component:



Next set up the composer.json:



\****psr-4*** is the latest PHP standard (PHP recommended standard for modern php apps), ***Acme*** is a fictitious name for our project and ***src*** specifies the source for our classes;

Namespaces are like folders of our app (they label and organize classes);

We have declared the Acme namespace in the composer.json and in the classes and are now using it. In order to set the autoloading we need to run ***composer dump-autoload*** (whenever we are modifying composer.json we need to run this command);

If we set up composer.json as (we indicate that we want psr-4 autoloading). As the key, we specify the root namespace (***Acme*** in this case):

{

    "autoload": {

        "psr-4": {

            "Acme\\": "src"

        }

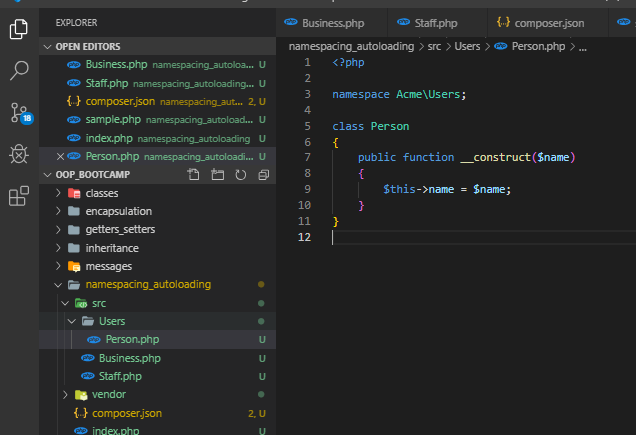
    }

}

\*we name the namespace by the product/app name and the “src” is the root directory for the namespace (the directory associated to the root namespace);

\*this means that any class we place in the root directory (src in this case) will have the Acme namespace;

\*also if we create other sub-directorie in the root directory (src), we have to reference the namespace as per below (note now Person has the Acme\Users namespace):



Furthermore, when we reference Person in a different class, we have to reference its namespace as well:

<?php

namespace Acme;

use Acme\Users\Person;

//we can typehint a class that is not part of the current namespace by preceding it with \ -> \Person or we can use the syntax above;

class Business

{

    protected $staff = [];

    public function \_\_construct(Staff $staff)

    {

        $this->staff = $staff;

    }

    //type hinting -> hire() can only be used with a Person object

    public function hire(Person $person)

    {

        //add person to the staff

        $this->staff->add($person);

    }

    public function getStaffMembers()

    {

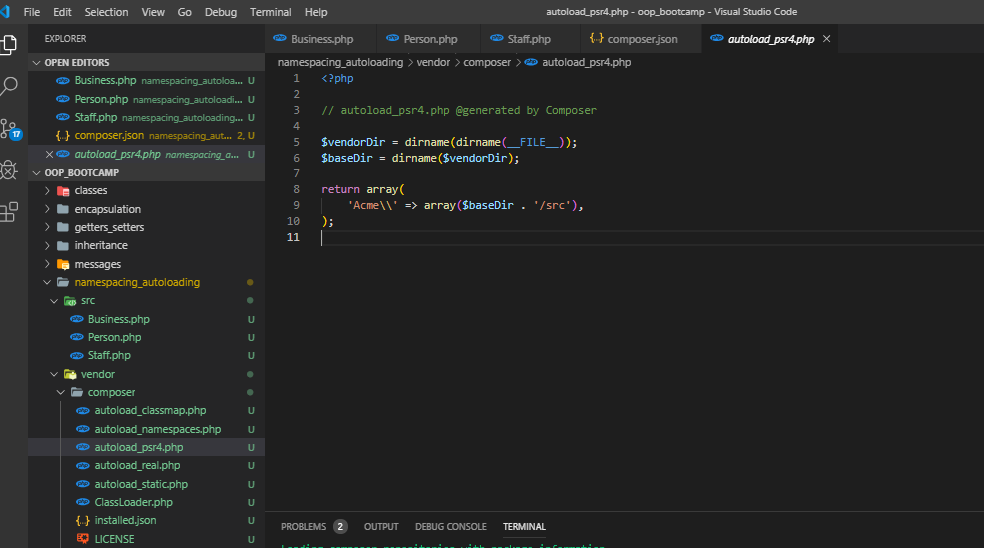
        return $this->staff->members();

    }

}

\*note how we use the Acme\Users\Person (we reference the Acme\Users namespace);

And next we run the command ***composer dump-autoload***, we get the autoload\_psr4.php file created in composer directory:



\*this shows us that it loads all the classes in the ‘src’ directory;

Now in index.php we have to require the autoloader and to modify the classes so that they correspond to the Acme namespace:

<?php

require 'vendor/autoload.php';

//create person

$jeffrey = new Acme\Person('Jeffrey Way');

//pass $jeffrey as param when instantiating the Staff class (intended behaviour expressed in the Staff constructor method)

$staff = new Acme\Staff($jeffrey);

$laracasts = new Acme\Business($staff);

$laracasts->hire(new Acme\Person('Jane Doe'));

// var\_dump($staff); die();

//replaced the simple staff dumping with a method of Business class which gets all the staff members;

var\_dump($laracasts->getStaffMembers());

\*now the script will work;

We can also include the namespaces as follows:

<?php

use Acme\Person;

use Acme\Business;

use Acme\Staff;

//create person

$jeffrey = new Person('Jeffrey Way');

//pass $jeffrey as param when instantiating the Staff class (intended behaviour expressed in the Staff constructor method)

$staff = new Staff($jeffrey);

$laracasts = new Business($staff);

$laracasts->hire(new Person('Jane Doe'));

// var\_dump($staff); die();

//replaced the simple staff dumping with a method of Business class which gets all the staff members;

var\_dump($laracasts->getStaffMembers());

\*now we don’t have to say new Acme\Person, new Acme\Business, etc…