**Composer**

# O que é e para que serve?

O PHP é uma das linguagens de programação mais usadas no mundo, isso ninguém pode negar. Mas, em relação a outras, tais como Ruby, Python e Perl, PHP possui uma tremenda desvantagem: não possui um sistema totalmente confiável e unificado para gerenciamento de pacotes. Ruby tem Gems, Python, PIP; Perl, CPAN; e, estendendo um pouco, é possível citar o [NPM](https://npmjs.org/) do Node.js. Mas e o PHP?

Na verdade, o PHP até tem um sistema de pacotes, o PEAR, mas, convenhamos, trabalhar com o PEAR não é algo divertido ou produtivo

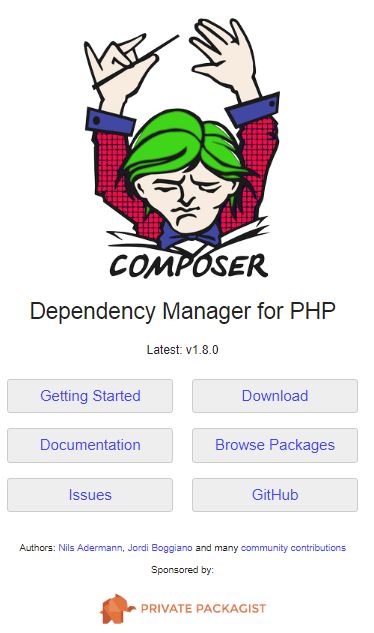
Composer é uma ferramenta para gerenciamento de dependências em PHP. Ele permite que você declare as bibliotecas dependentes que seu projeto precisa e as instala para você

Usando Composer, você simplesmente especifica quais pacotes (códigos reutilizáveis) seu projeto precisa – podendo estes pacotes também ter dependências – e ele vai, automaticamente, baixar isso e incluir nos locais apropriados de seu projeto!

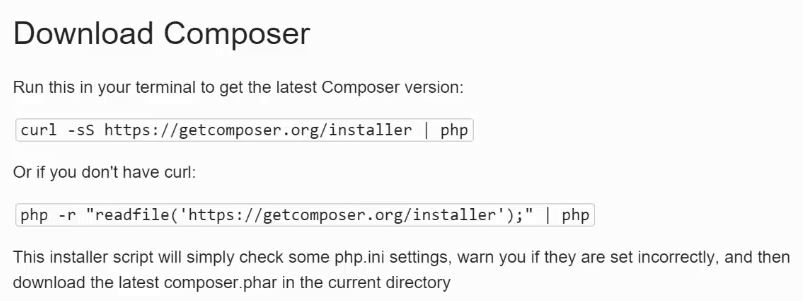
Caso seja preciso acrescentar, remover ou atualizar algum pacote, sem problemas: o gerenciador também faz o trabalho todo! Acreditem: depois que se começa a trabalhar usando este tipo de ferramenta, realmente é difícil abrir mão de suas facilidades e comodidades; além disso, a produtividade vai às alturas.

# Download

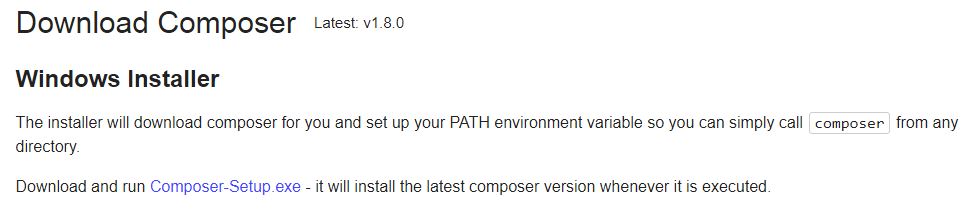
<https://getcomposer.org>



Clique no botão "*Download*".



No caso de estar usando Windows, basta fazer o download do arquivo executável Composer-Setup.exe:



# Como funciona o Composer



O Composer funciona, basicamente, através de duas vertentes: um **repositório** para os pacotes (Packagist) e **instruções via linha de comando** para gerenciamento dos pacotes (para procurar, instalar, atualizar, remover, etc).

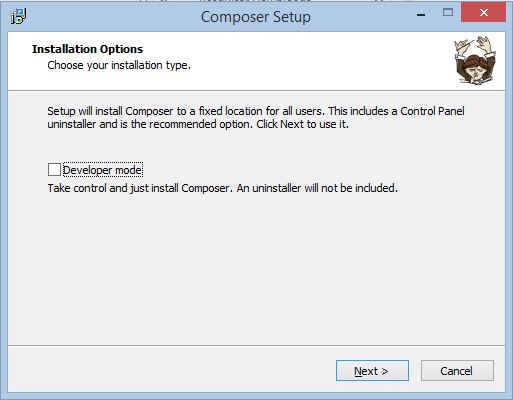
A instalação dos pacotes é feita *por projeto* e, *por default*, nada é instalado globalmente. Por isso o Composer é considerado mais um **Gerenciador de Dependências** do que um *Gerenciador de Pacotes* (mas usar o termo “pacotes”, no caso, também não é errado).

Depois de configurar corretamente o arquivo composer.json para instalar as dependências informadas – através do comando install, no Terminal – é feita uma verificação para ver se há algum erro de sintaxe, acontece a busca no repositório pelo pacote informado, o download é realizado e a “instalação” feita no diretório apropriado – por padrão, é o diretório vendor, na raiz do projeto, mas isso também é configurável.

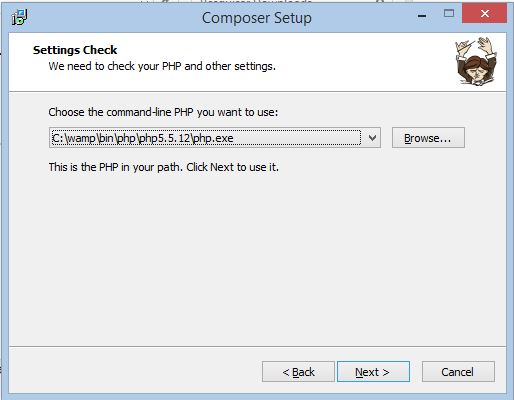
O primeiro passo para entrar no mundo de gerenciamento de pacotes PHP é instalar o Composer.

# Instalação

Para instalar o Composer, antes, instale o Wamp. Após o download do arquivo Composer-Setup.exe, clique nele para iniciar a instalação:

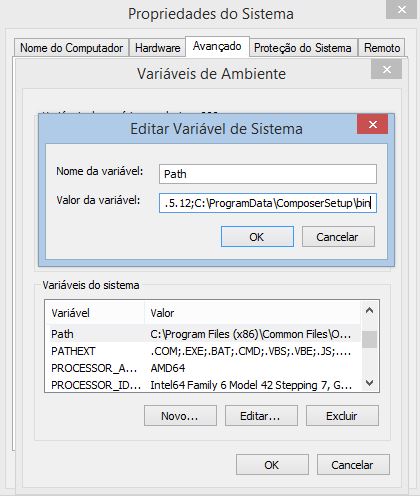


Clique no botão "Next".



Clique novamente no botão "Next". Na tela que se abrir, clique novamente no botão "Next" e finalmente clique no botão "Install" e confirme os demais passos até finalizar a instalação.

Após a instalação, reinicie o computador. Confirme, em "variáveis de ambiente" do Windows, se no path foi incluído o caminho para o Composer:



Na pasta c:\wamp\www crie a pasta "curso\_composer".

No prompt do DOS (cmd), acesse a pasta criada e entre com o comando:

C:\wamp\www\curso\_composer> composer

Se estiver tudo ok, será apresentada a tela a seguir:



# Iniciando o Composer

Ainda na pasta criada, entre com o comando:

composer init

* Como nome do pacote entre com: eti/composer-php
* Em "Description" entre com: Composer Curso de PHP
* Em "author" entre com "Carlos Ferreira <carlos@especializati.com.br>"
* Em "Minimum Stability", "Package Type" e "License", tecle "Enter".
* Nas próximas duas perguntas entre com "no".



Na última pergunta confirme, digitando "yes" a geração do arquivo. O arquivo composer.json será criado.

# Arquivo composer.json

Um dos principais arquivos para se trabalhar com Composer é o composer.json. É nele que as instruções sobre os pacotes que serão usados no projeto ficam contidas. Este é um arquivo “.json” comum que deve ficar na raiz de seu projeto. Existe uma gama de parâmetros possíveis para o gerenciamento de dependências, mas o mais usado, certamente, é o require.

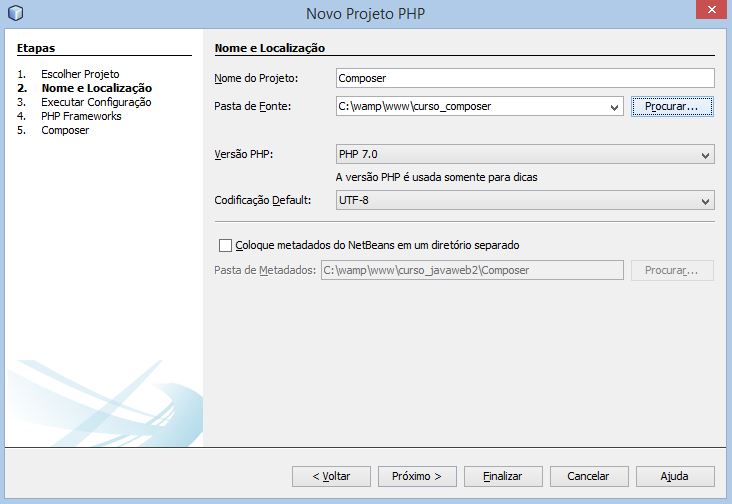
A diretiva “require” no composer.json informa ao Composer quais os pacotes necessários para o projeto. Como citado, o repositório central é o Packagist (https://packagist.org/), então, apesar de ser possível trabalhar com outras “fontes de pacotes”, por padrão é lá que os pacotes que você vai usar estão.

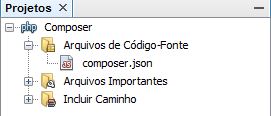
# Arquivo composer.lock

Depois de instalar as dependências informadas, o Composer vai criar automaticamente, também na raiz do projeto, o arquivo composer.lock, com informações de quais pacotes estão instalados e quais as versões destes pacotes.

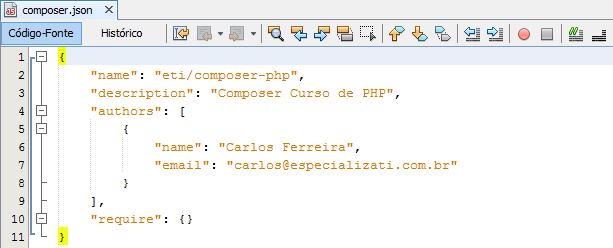
# Trabalhando no NetBeans

Abra o NetBeans e crie um novo projeto (Aplicação PHP), na pasta criada (curso\_composer), com o nome de "Composer".





Conteúdo do arquivo composer.json:



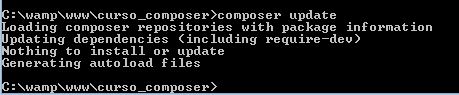
* O “name” é o nome de sua aplicação. Esta marcação é opcional mas recomendada.
* O “description” é uma breve descrição do que sua aplicação se propõe a fazer. Também opcional.
* Em “authors” aparecem os créditos de desenvolvedores que contribuiram com o projeto.

O “require” basicamente deixa claro quais são as dependências de sua aplicação. Neste caso se a versão do PHP for abaixo da 5.2.8 simplesmente uma mensagem de erro será lançada ao instalar as dependências lhe informando que não é possível prosseguir por nem todos os requisitos estarem satisfeitos.

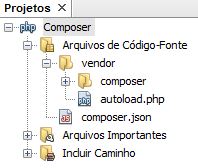
# Atualizando pacotes

A atualização de pacotes é feita com o comando composer update. Sem fazer nenhuma alteração, na pasta do projeto, entre com o seguinte comando:

C:\wamp\www\curso\_composer> composer update



Foi informado que não foi feita nenhuma instalação ou atualização.



Foi criada uma pasta chamada "vendor" contendo uma subpasta chamada "composer" e um arquivo chamado autoload.php.

Todos os pacotes de terceiros instalados ficarão dentro da pasta "vendor".

# Arquivo autoload.php

Então, depois de corretamente instalados os pacotes e o arquivo composer.lock ser gerado, é preciso “ensinar” ao seu projeto como autocarregar as dependências para que sejam usadas quando necessário.

# Arquivo index.php

Agora já temos tudo. O composer gerenciando as dependências, as dependências definidas em nosso arquivo composer.json, e uma pasta contendo todas as dependências necessárias juntamente com o autoloader do composer que encarrega-se de registar todos os namespaces dos arquivos baixados na pasta vendor. Com isso basta utilizarmos.

Dentro da pasta raiz, crie um arquivo chamado index.php e inclua o autoloader do composer conforme o exemplo abaixo.

  <?php

header('Content-Type: text/html; charset=utf-8');

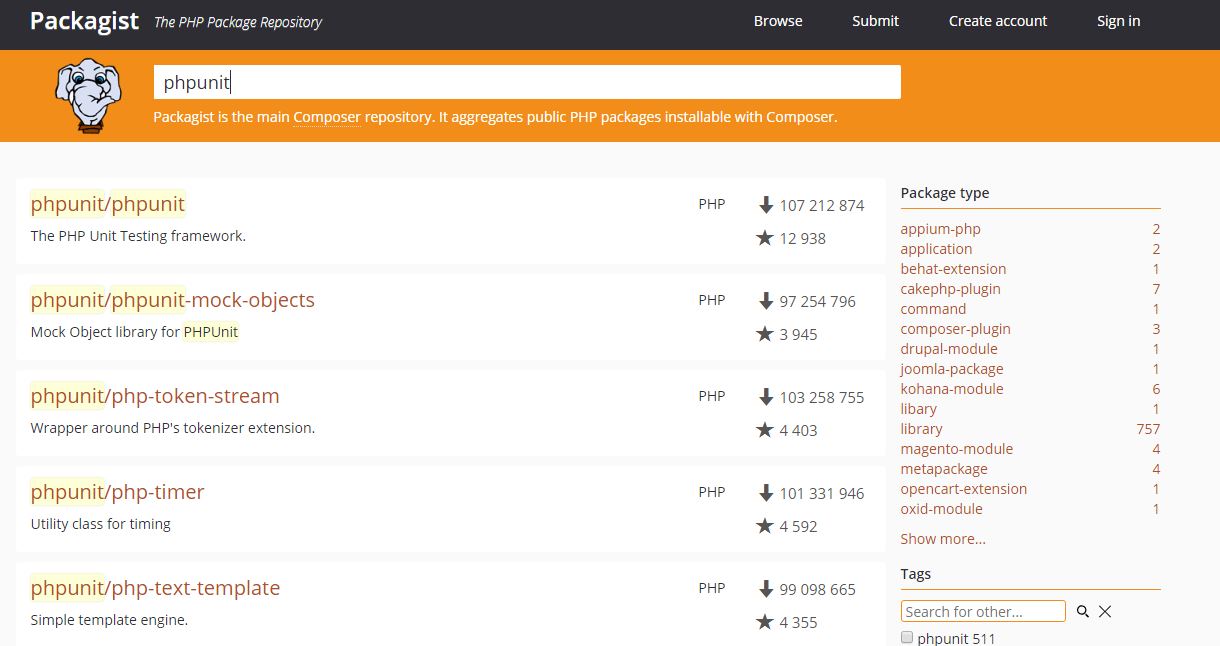
require\_once './vendor/autoload.php';

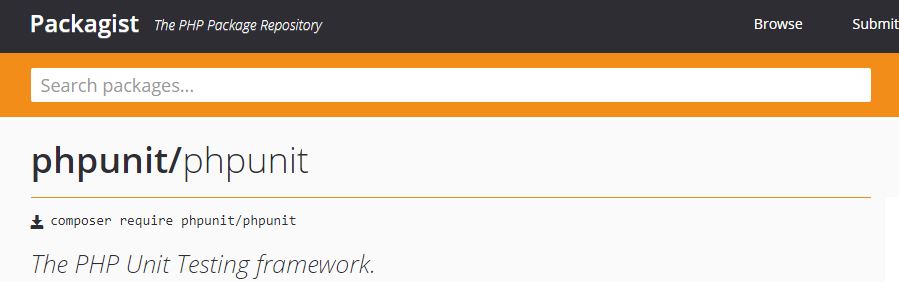
# Escolhendo pacotes no Packagist

Agora que você já tem o esqueleto de seu composer configurado em sua aplicação falta incluir alguns pacotes. O composer utiliza como seu repositório o Packagist (<https://packagist.org/>) onde qualquer desenvolvedor pode criar seus próprios pacotes e disponibilizá-los para a comunidade semelhante o github (https://github.com/). O Packagist lhe fornece o total de instalações dos pacotes por dia, mês e o total. O mais legal é que estas estatísticas são fiéis, ou seja, se alguém remover um pacote do seu \_composer.json\_ o total de instalações é reduzido. Com esta informação restam contagens apenas aplicações que realmente estão utilizando determinado pacote.

# Instalando o pacote PHPUnit

O PHPUnit é utilizado para fazer testes de software.





Altere o arquivo composer.json para:

{

"name": "eti/composer-php",

"description": "Composer Curso de PHP",

"authors": [

{

"name": "Carlos Ferreira",

"email": "carlos@especializati.com.br"

}

],

"require": {

"php":">=5.3.3"

},

"require-dev":{

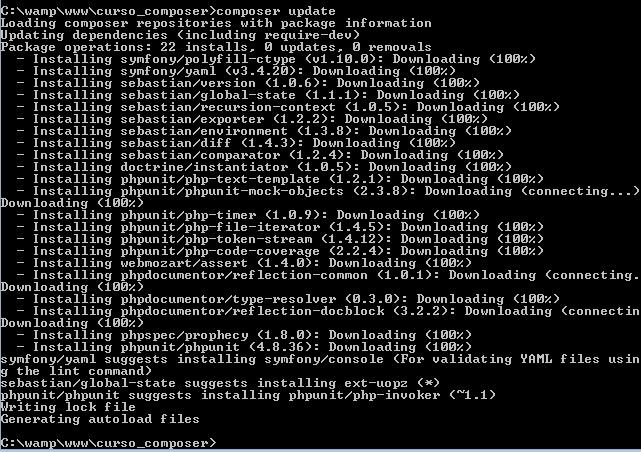
"phpunit/phpunit": "^4.3"

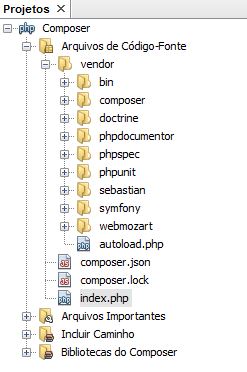
}

}

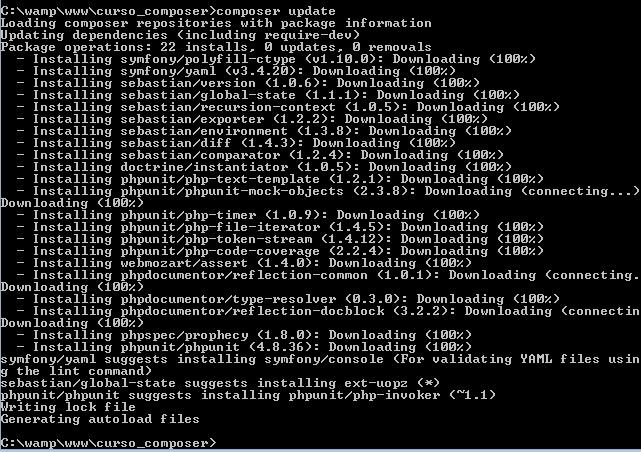
No prompt do DOS, na pasta do projeto, entre com o seguinte comando:

C:\wamp\www\curso\_composer> composer update

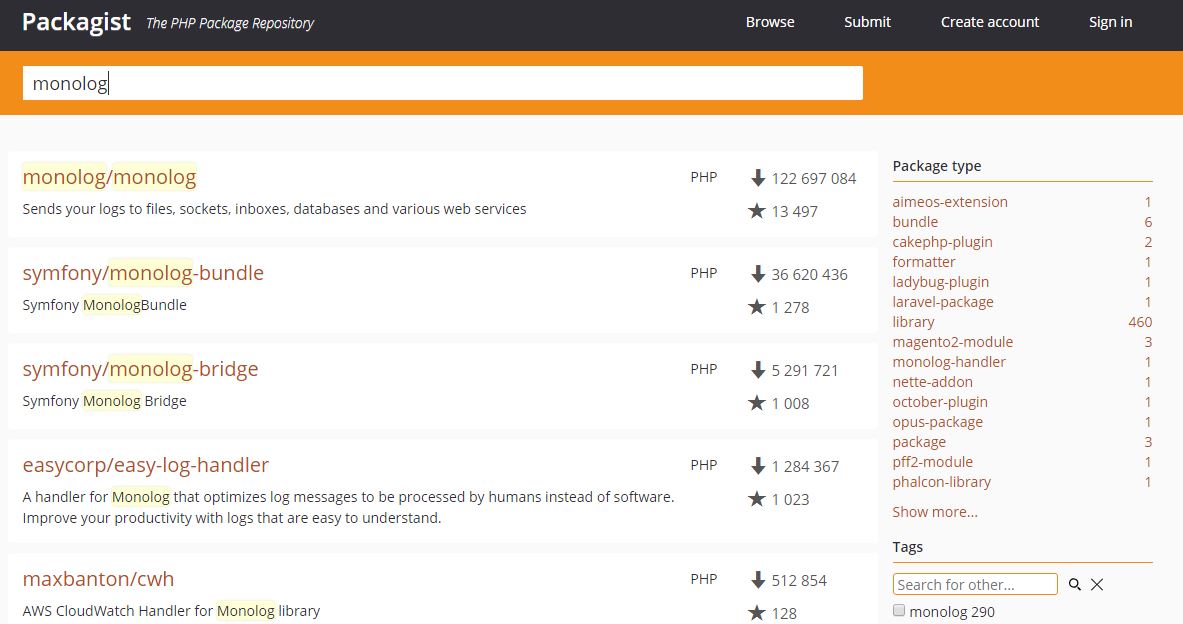


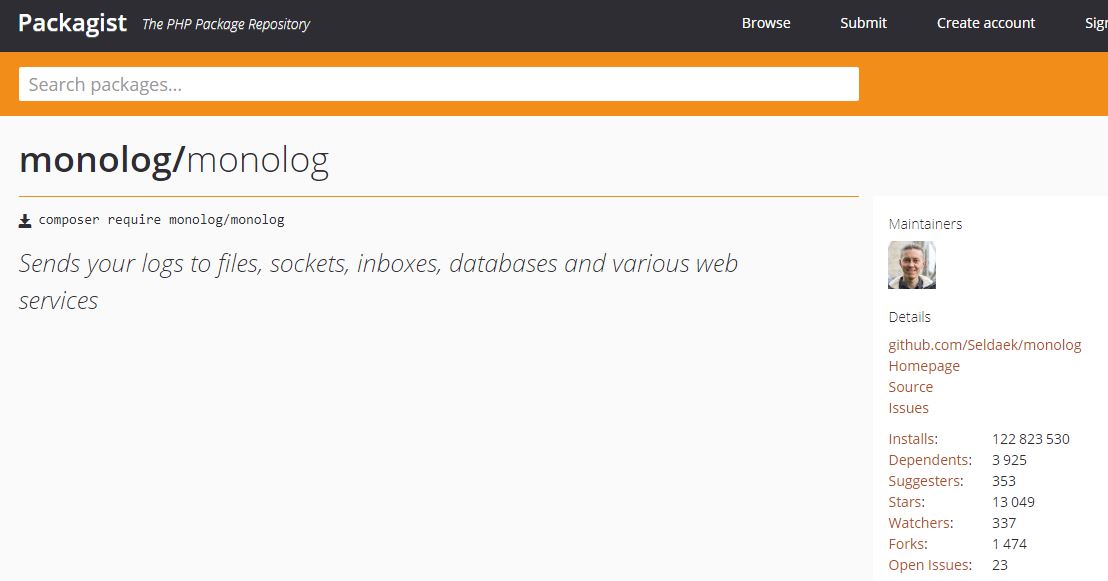


Serão instalados as dependências do pacote PHPUnit dentro da pasta "vendor".



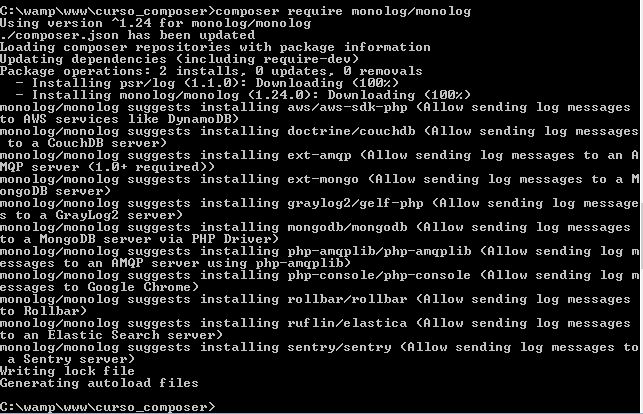
# Instalando o pacote monolog





Na pasta do projeto, entre com o seguinte comando:

C:\wamp\www\curso\_composer> composer require monolog/monolog



O conteúdo do arquivo composer.json será alterado para:

{

"name": "eti/composer-php",

"description": "Composer Curso de PHP",

"authors": [

{

"name": "Carlos Ferreira",

"email": "carlos@especializati.com.br"

}

],

"require": {

"php":">=5.3.3",

"monolog/monolog": "^1.24"

},

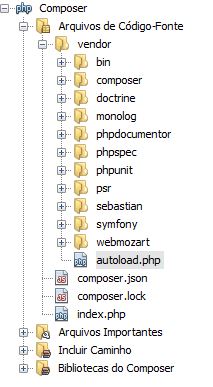
"require-dev":{

"phpunit/phpunit": "^4.3"

}

}

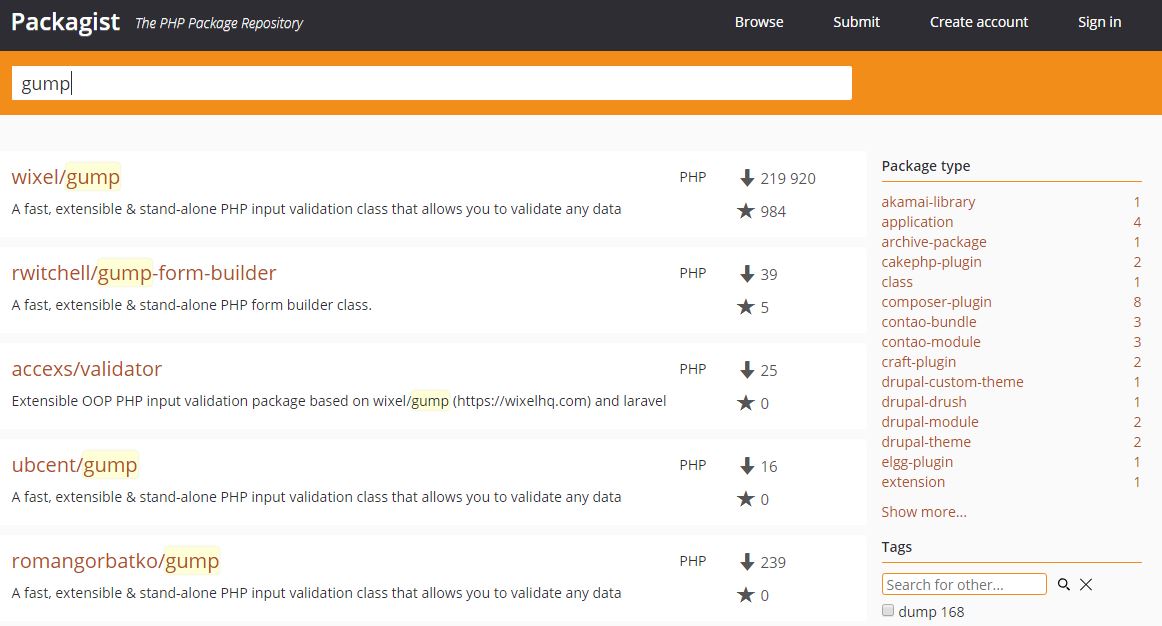
Na pasta vendor será incluída uma pasta para o pacote monolog:

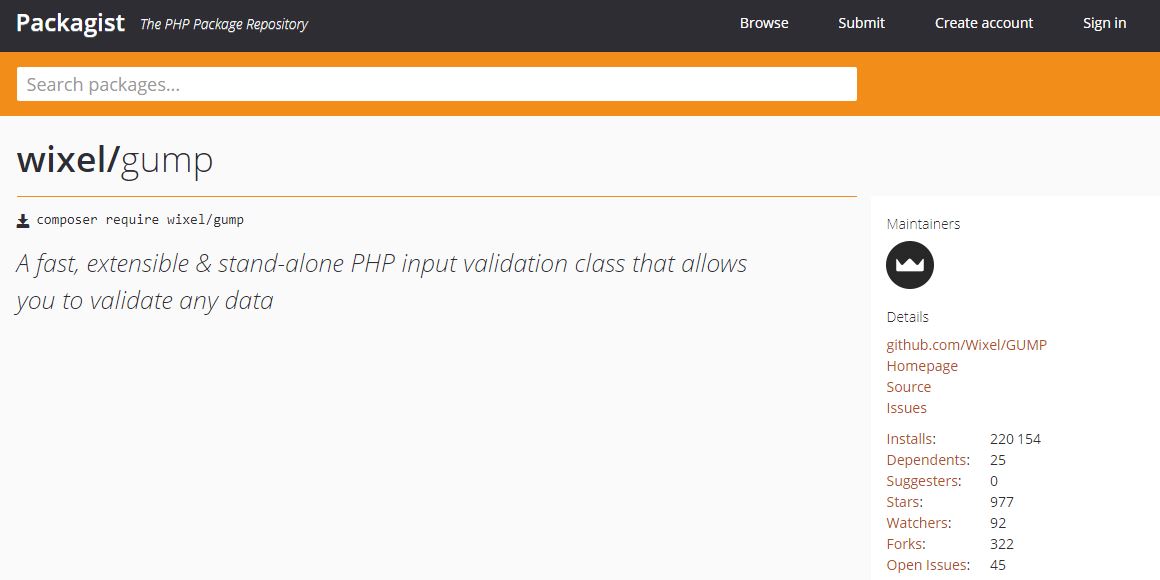


Para usar o monolog basta consultar sua documentação.

# Instalando o pacote gump

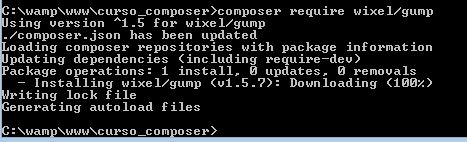
É utilizado para validação de dados.





Na pasta do projeto, entre com o seguinte comando:

C:\wamp\www\curso\_composer> composer require witzel/gump



O conteúdo do arquivo composer.json será alterado para:

{

"name": "eti/composer-php",

"description": "Composer Curso de PHP",

"authors": [

{

"name": "Carlos Ferreira",

"email": "carlos@especializati.com.br"

}

],

"require": {

"php":">=5.3.3",

"monolog/monolog": "^1.24",

"wixel/gump": "^1.5"

},

"require-dev":{

"phpunit/phpunit": "^4.3"

}

}

# Usando o pacote

Altere o arquivo index.php para:

<?php

header('Content-Type: text/html; charset=utf-8');

use \GUMP;

require\_once './vendor/autoload.php';

$dadosForm = ['nome' => 'Carlos Ferreira', 'email' => 'carlos@especializati.com.br'];

$regras = [

'nome' => 'required|max\_len,100|min\_len,3',

'email' => 'required|max\_len,150|valid\_email'

];

$is\_valid = GUMP::is\_valid($dadosForm, $regras);

if($is\_valid === true) {

echo "Cadastrar dados, correto!";

} else {

print\_r($is\_valid);

}

