Aula 5º

Sistemas de controle de dependências

Olá, alunos(as),

Bem-vindos(as) a nossa quinta aula da disciplina "Gerência de Configuração". Nesta aula, vamos saber como funcionam os sistemas de controle de dependências de software. Você vai aprender como funcionam esses sistemas que apoiam a Gerência de Configuração. Além disso, vamos ver como funcionam os gerenciadores NPM e Composer.

Assim, não se esqueçam de ler esta aula, assistir aos nossos vídeos e fazer as atividades. Se tiverem alguma dúvida, vocês podem usar o nosso quadro de avisos, que eu responderei.

Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- saber como funcionam os sistemas de controle de dependências;
- entender a necessidade de se usar um gerenciador de dependências;
- conhecer como funciona o gerenciador de pacotes Composer.

Seções de estudo

1 – Sistemas de controle de dependências
2 – Gerenciador de pacotes composer

1 - Sistemas de controle de dependências

Quando estamos em um projeto de desenvolvimento de software, muitas das vezes não precisamos recriar tudo do zero. Podemos usar bibliotecas que podem facilitar em nossa tarefa de desenvolvimento de software, tais como:

- Bibliotecas que oferecem novas funcionalidades e recursos que podem acelerar o desenvolvimento de software, como o **JQuery** e os drivers de conexão a SGBDs para JDBC;
- Frameworks que nos oferecem uma base mínima e uma gama de recursos que pode ser utilizada para acelerar o desenvolvimento de software, como o **Laravel, CakePHP e Express**;
- Bibliotecas que oferecem componentes visuais, que podem incrementar a usabilidade do software como, por exemplo, um componente de seleção de data e hora.

Quando usamos uma biblioteca pronta para o nosso projeto de software, ela passa a ser chamada de **dependência**, pois o nosso *software* passa a depender da nossa biblioteca para que possa funcionar adequadamente. (OMBREDANNE, 2016)

Muitas dependências podem ser gerenciadas de maneira manual, naqual podemos fazer o download dos arquivos dessa dependência (que podem vir de maneira compilada ou em forma de códigos-fonte), salvar esses arquivos na pasta do projeto e depois incorporar essas dependências ao nosso arquivo fonte. Isso é feito por meio de comandos de requisição, tais como:

- os comandos include e require do PHP;
- o comando import da linguagem Java;
- as tags style (para a incorporação de arquivos CSS) e script (para a incorporação de arquivos-fonte JavaScript) em arquivos HTML.

Porém, isso pode gerar alguns problemas. Vamos agora apresentar uma breve história para ilustrarmos esses problemas:

Hoje em dia, projetos web tendem a utilizar muitos componentes prontos, existem frameworks, plugins, assets para animação, para estilo, elementos para formulários, para notificações... E estas dependências são muito importantes para agilizar o desenvolvimento de software e mantê-lo simples além de não termos que nos preocupar em "reinventar a roda".
Durante o desenvolvimento de um projeto vamos adicionando novos componentes para atender nossas necessidades.

"Humm... Estou precisando de um elemento de formulário para que o usuário escolha uma data. Ahaa vou utilizar este datepicker v3.2.1 baixando o arquivo datepicker.min.js e colocando nesta pasta libs/."
"Vou utilizar o Bootstrap 3 como framework css principal e vou adicionar também o jQuery 2.10.1 que facilita muito minha vida."

E com novos desenvolvedores no projeto pode ser que eles prefiram utilizar outras dependências e assim vão adicionando.
Logo vemos que o projeto possui muitas dependências e de certa forma isto não é ruim, mas gerenciá-las se torna complicado.

"Saiu uma nova versão do jQuery e as novas funcionalidades são muito boas! Como eu posso atualizo?"
"É meio complicado atualizar pois o datepicker depende da versão 2 e não sei se o novo jQuery vai suporta-la. É preciso atualizar o datepicker antes"

Fonte: VIEIRA, 2016

Como você pode ver, o gerenciamento manual de dependências pode causar alguns problemas para o progresso do sistema, principalmente na hora de atualizar as dependências, visto que uma dependência pode acabar sendo incompatível com outra dependência da qual ela depende. Essas incompatibilidades podem causar quebra no sistema.

A essa altura, você deve estar se perguntando: como podemos fazer um gerenciamento das dependências de maneira eficiente?

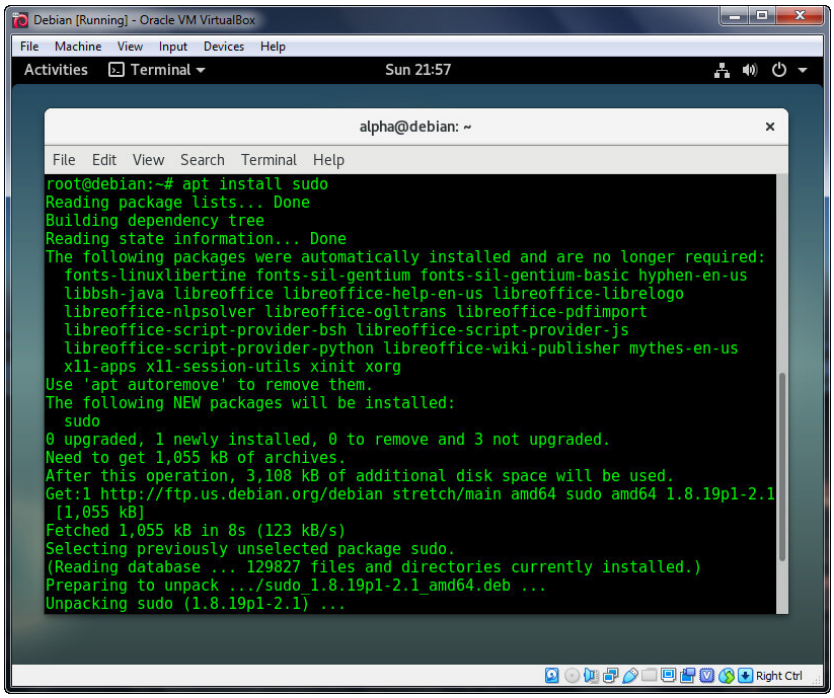
Para resolver esse problema, surgiram os gerenciadores de dependência. Eles foram inspirados nos gerenciadores de pacotes das distribuições Linux, que agilizam a instalação e a atualização dos programas do sistema e de suas dependências. Um dos exemplos mais famosos é o utilitário **APT**, que é utilizado na distribuição Debian Linux e suas derivadas, como o Ubuntu e o Linux Mint.

Para fazer a instalação de um programa do sistema nesses sistemas equipados com esse utilitário, basta apenas usar – em um terminal com privilégios de superusuário – o comando **apt install**, junto com o nome do programa que deseja instalar. Por exemplo, se precisássemos instalar em uma máquina Debian o SGBD PostgreSQL, poderemos usar o seguinte comando:

```
apt install postgresql-11
```

O APT se encarrega de instalar o pacote (assim denominado os programas e bibliotecas que podem ser instalados por meio dessa ferramenta) e, em seguida, instalar os pacotes do qual esse programa depende, de maneira automatizada.

Figura 1 – Exemplo de uso da ferramenta APT.



Fonte: Disponível em: https://tutorialsoverflow.com/how-to-fix-sudo-command-not-found-in-debian/. Acesso em 28 out. 2019.

E quando não precisarmos usar o pacote? Basta usar o seguinte comando:

```
apt-get remove postgresql-11
```

Esse comando faz a desinstalação do pacote de maneira automatizada, sem a necessidade de remover pastas ou editar arquivos de configuração manualmente.

Esse é o mesmo raciocínio que é aplicado nos gerenciadores de dependências. Há um comando simples de instalação e remoção de pacotes, que são salvos em uma pasta predeterminada pelo gerenciador de pacotes, localizada dentro da pasta do projeto. Quando uma instalação é solicitada, o gerenciador verifica quais são as dependências para o pacote e os instala.

Mas os gerenciadores de dependências têm algo a mais. Para permitir que todas as dependências possam ser instaladas de uma só vez, em suas versões corretas, os gerenciadores permitem que a lista de dependências seja salva em um arquivo, para que sejam listadas todas as suas versões em seus lugares corretos, além de outras configurações.

A seguir, apresentamos um exemplo de arquivo de listagem de dependências, gerado pelo Composer, para um projeto hipotético. Perceba o objeto "require", a qual indica claramente as dependências, junto com as suas respectivas versões.

```
{
    "name": "aureliocasoni/teste",
    "description": "\"Site Teste\"",
    "type": "site",
    "require": {
        "twig/twig": "~2.0",
        "twig/extensions": "^1.4",
        "phpmailer/phpmailer": "^5.2",
        "kevinlebrun/slug.php": "^1.0",
        "jralph/twig-markdown": "^2.0",
        "michelf/php-markdown": "~1.3"
    },
    "license": "proprietary",
    "authors": [
        {
            "name": "Luís Aurélio Casoni",
            "email": "luis@a.com"
        }
    ],
    "autoload": {
        "psr-4": {
            "App\\": "app/"
        }
    }
}
```

Perceba que esse arquivo contém outros dados do projeto como, por exemplo, o nome do projeto, o autor do projeto, entre outros dados. Além disso, como descreve um **conjunto de versões de componentes de software**,

esse arquivo pode ser definido como a **descrição de uma *baseline* de um sistema**, de acordo com a definição de SOMMERVILLE (2011).

A seguir, vamos apresentar como funciona o gerenciador de pacote Composer.

2 - Gerenciador de pacotes composer

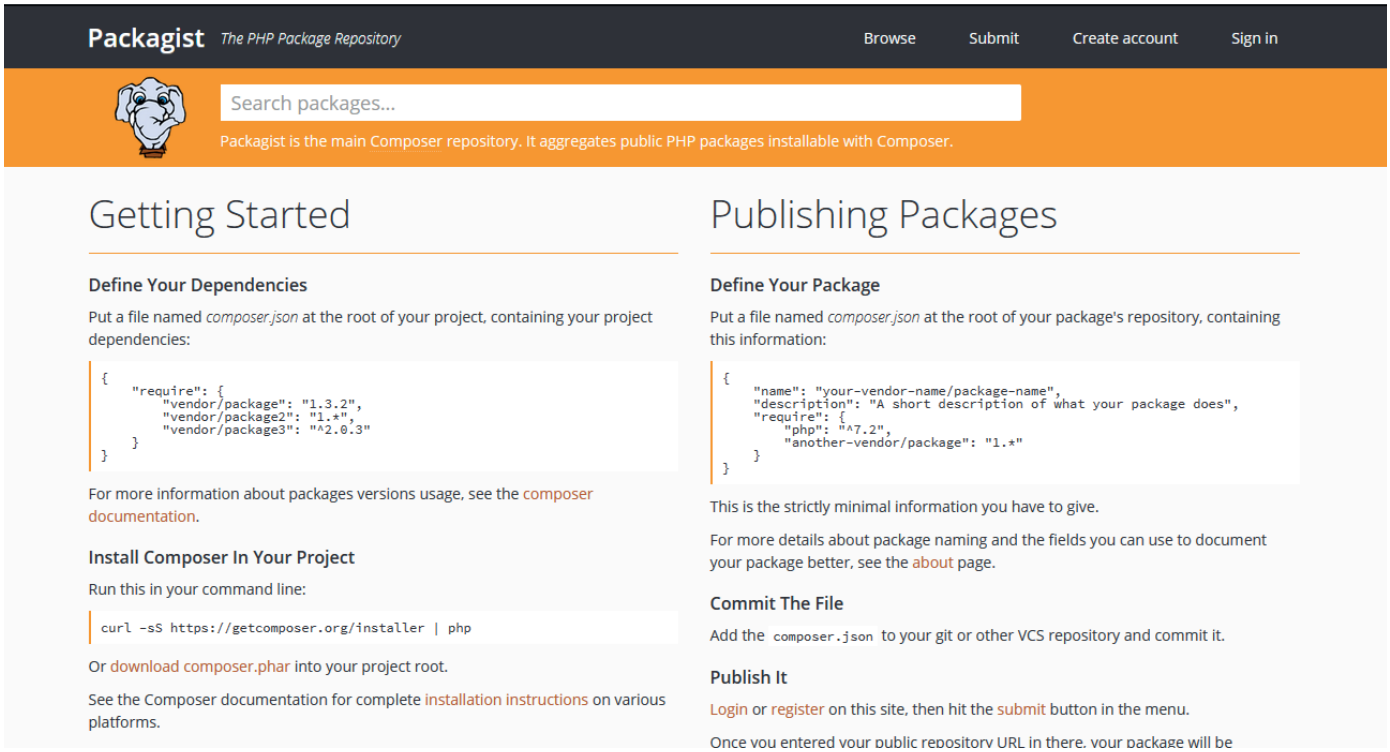
O Gerenciador de Pacotes Composer é um gerenciador criado para lidar com bibliotecas escritas na linguagem PHP. Criado por Nils Adermann e Jordi Boggiano, o utilitário foi escrito na linguagem PHP, requerendo que o interpretador PHP esteja instalado na máquina.

Figura 2 – Logomarca oficial do Composer, criada por WizardCat.

Fonte: Disponível em: https://commons.wikimedia.org/wiki/File:Logo-composer-transparent.png. Acesso em 28 out. 2019. This file is licensed under the Expat License, sometimes known as the MIT License

Os pacotes que o Composer gerencia estão listados em repositórios, sendo que o principal deles é o repositório público Packagist. Seu site oficial (https://packagist.org/) permite a consulta dos pacotes pelo nome. Além disso, a página permite que o usuário consulte informações importantes do pacote, tais como: a versão atual, as dependências necessárias, o nome do autor, as instruções de instalação, entre outros.

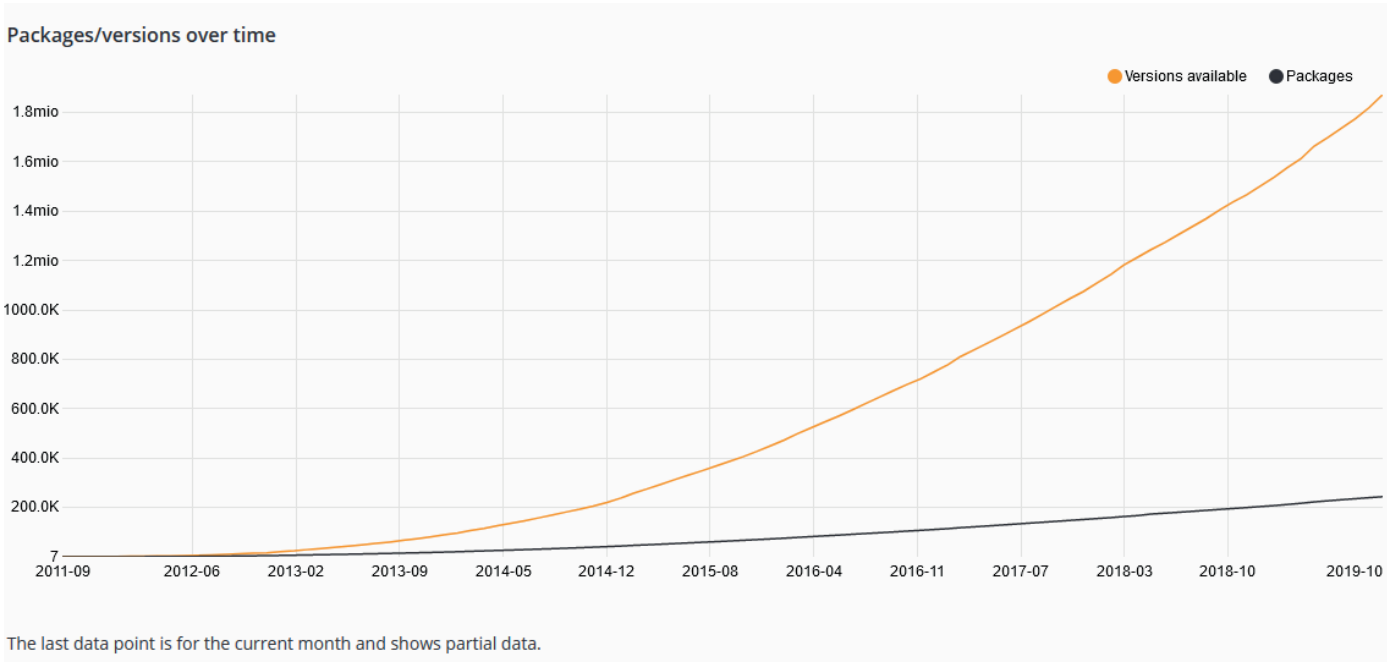
Figura 3 – Site oficial do Packagist, em 28 out. 2019.



Fonte: Imagem do acervo pessoal extraído do site: https://packagist.org/ em 28 out. 2019.

O Packagist existe desde 2011. De lá para cá, ganhou muita adoção pela comunidade do PHP. Segundo a sua página oficial de estatísticas (https://packagist.org/statistics), em outubro de 2019, existiam mais de 240 mil pacotes registrados na plataforma e quase 2 milhões de versões registradas. Em dias de pico, o repositório recebe mais de 30 milhões de requisições de instalação de pacotes em um dia.

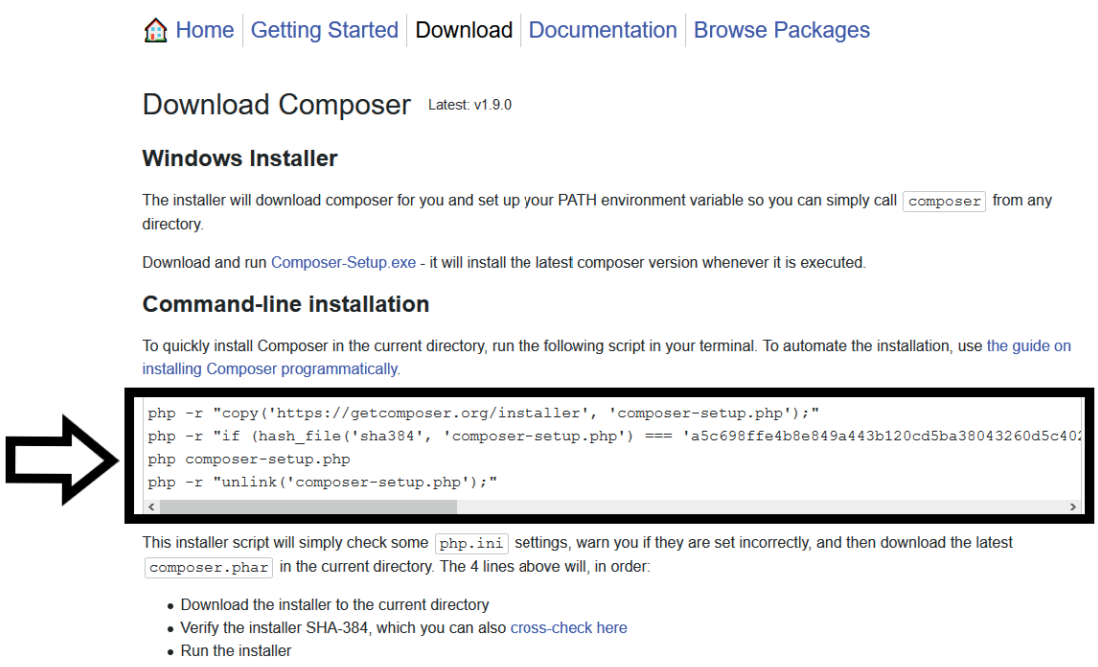
Figura 4 – Gráfico que mostra o progresso do número de pacotes registrados no Packagist e o número de versões registradas.



Fonte: Disponível em: https://packagist.org/statistics. Acesso em 28 out. 2019.

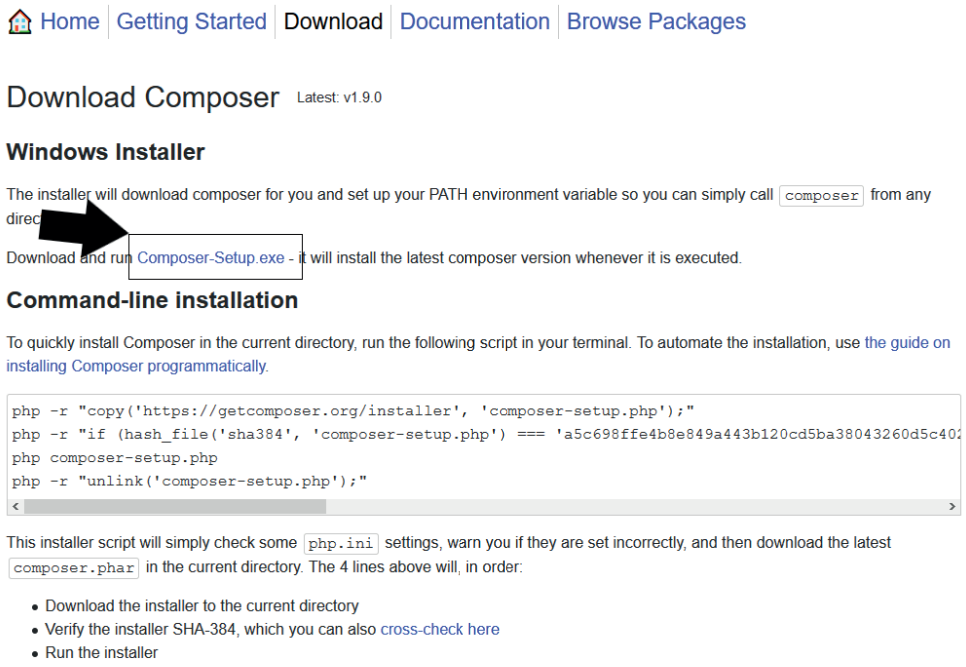
Para instalar o Composer, há duas opções, sendo que a escolha depende de cada sistema. Se o sistema for Linux, basta apenas seguir os seguintes passos:

1. Instalar o PHP – isso depende da distribuição do Linux a qual você está usando;
2. Abrir um terminal com privilégios de superusuário;
3. Acessar o site: https://getcomposer.org/download/;
4. No site, procure a seção **Command-line installation**. Nessa seção, deve aparecer uma caixa com quatro linhas que começam com "php", como indica a imagem a seguir. Copie essas quatro linhas;

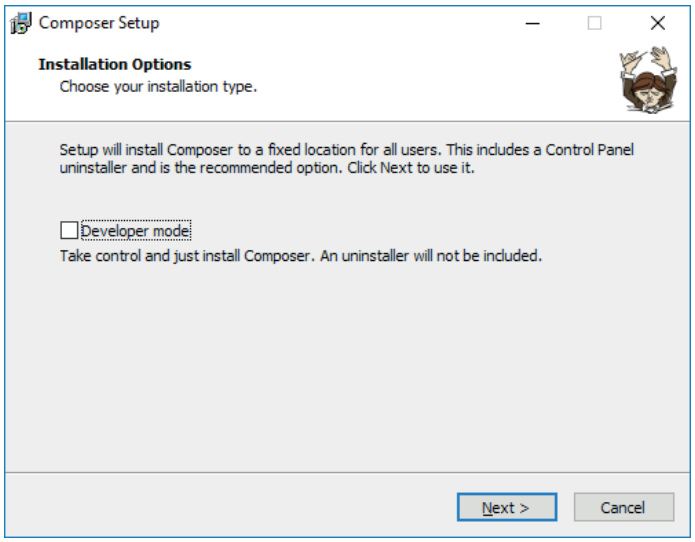


5. Cole as linhas no terminal e aguarde o fim do processo;
6. O Composer estará instalado.

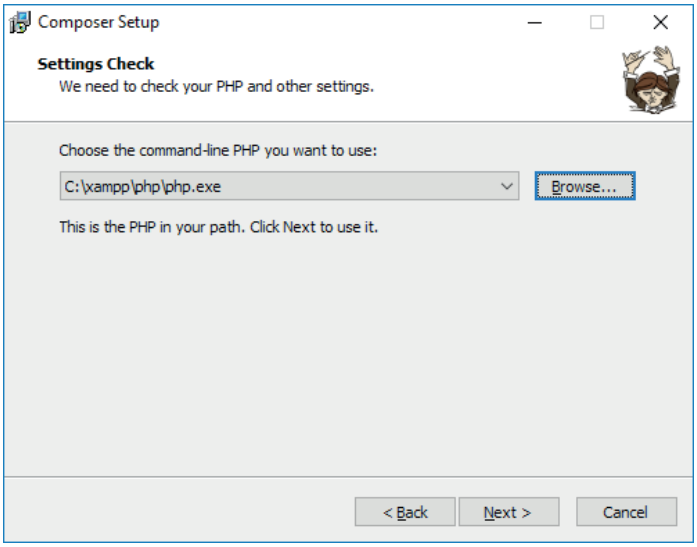
Caso o seu sistema seja Windows, basta você baixar o instalador do programa. Para baixar, acesse o site: https://getcomposer.org/download/. Procure pelo link azul, chamado de **Composer-Setup.exe** e clique nele. Siga as instruções do navegador para fazer o *download* do arquivo.



Ao terminar de fazer o *download*, execute o arquivo. Quando aparecer a primeira tela, clique em "Next":

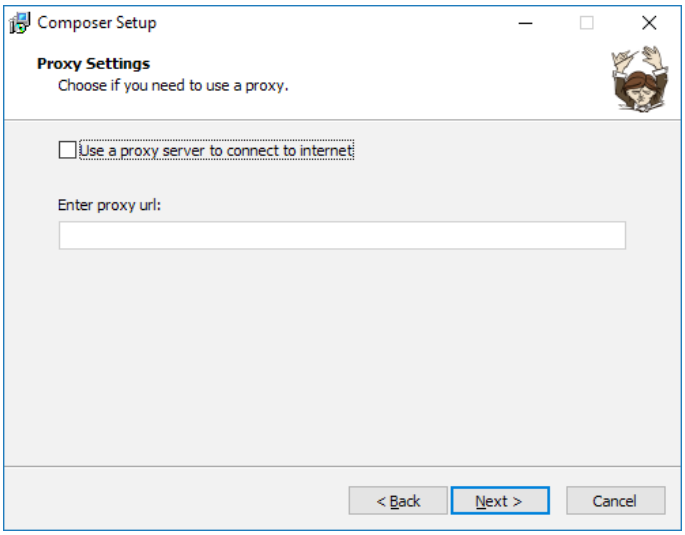


Na segunda tela, você deve selecionar em qual arquivo está o executável do PHP. Clique em "Browse" e selecione o arquivo que contém o executável do PHP em sua máquina. Ao final, clique em "Next".

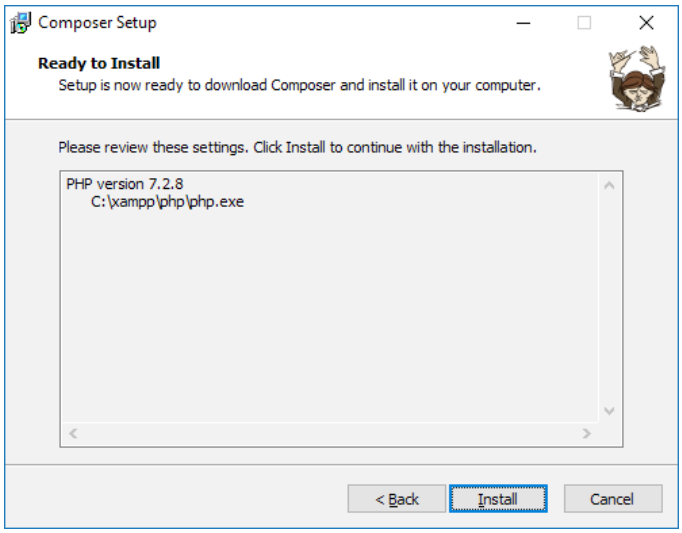


Se estiver usando o XAMPP, o executável estará no diretório "php" da pasta de instalação do XAMPP (normalmente é C:\xampp), chamado de "php.exe".

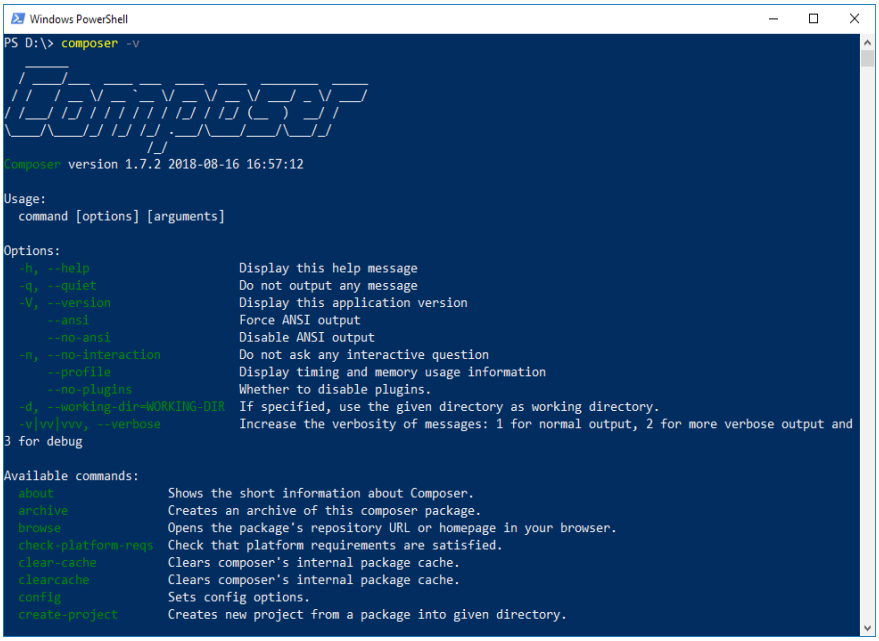
Na próxima tela, o instalador pedirá para indicar as configurações de proxy. Caso não tenha, basta apenas clicar em "Next".



Na tela a seguir, o Composer pergunta se deseja confirmar com a instalação. Para confirmar, clique em "Next":



Aguarde o fim da instalação e o Composer estará instalado no sistema! Para verificar, abra um terminal e digite o comando "composer" (sem as aspas). Você deve ver um resumo dos comandos do programa:

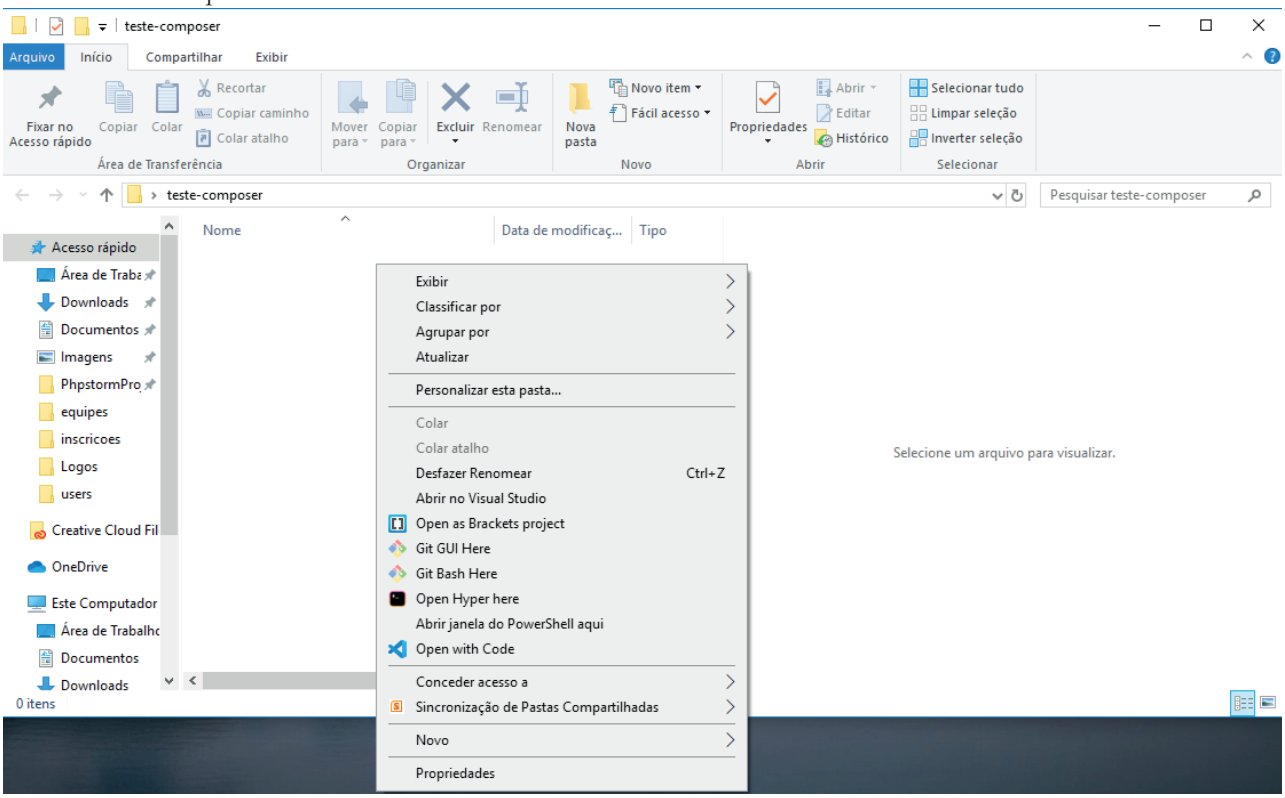


Agora que você sabe como funciona a instalação do Composer, vamos ver como funciona um workflow de uso. Vamos lá?

2.1 – Usando o Composer

Para usar o Composer, precisamos criar um novo projeto, em uma pasta dedicada para isso. Em um projeto da vida real, essa pasta seria a raiz dos arquivos de um site escrito na linguagem PHP.

Vamos criar uma pasta, denominada de "teste-composer", para abrigar os arquivos do nosso projeto. Em seguida, vamos abrir um terminal dentro da pasta "teste-composer". Para isso, com a pasta aberta, clique com o botão direito do mouse enquanto mantêm pressionado a tecla Shift. No menu pop-up que aparecer, clique em "Abrir janela do terminal aqui" ou "Abrir janela do PowerShell aqui":



Uma janela do terminal do Windows será aberta, configurado para o diretório do nosso projeto.

O próximo passo será inicializar o nosso projeto. Esse processo consiste na criação do arquivo composer.json, que serve para abrigar os dados do nosso projeto, em especial, a lista das dependências que serão usadas nesse projeto.

O arquivo que você viu na primeira seção que nós usamos para exemplificar é um arquivo composer.json.

Para fazer isso, devemos digitar o comando composer init. Um assistente será aberto, que fará as seguintes perguntas:

- Qual é o nome do pacote? Digite o nome do seu projeto;
- Qual a descrição do pacote? Pode deixar em branco se desejar;
- Qual é o autor do projeto? Se não quiser colocar, basta digitar **n** e teclar "Enter";
- Qual é a estabilidade mínima do pacote? Pode deixar em branco se não quiser especificar;
- Qual é o tipo do pacote? Digite o texto "project" e tecle Enter;
- Qual é a licença do seu projeto? Se você aderir a alguma licença open-source, digite o nome da licença correspondente. Se não tiver alguma licença adotada, pode deixar esse campo em branco.

```
PS D:\teste-composer> composer init

  Welcome to the Composer config generator


This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [[redacted]teste-composer]: luis/teste
Description []: teste
Author [Luis Aurelio Casoni <luis.aurelio12@gmail.com>, n to skip]: n
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []:
```

Depois, o Composer pergunta se você deseja definir as suas dependências de maneira interativa. Nesse caso, não vamos fazer isso. Digite "no" e tecle Enter para as duas perguntas:

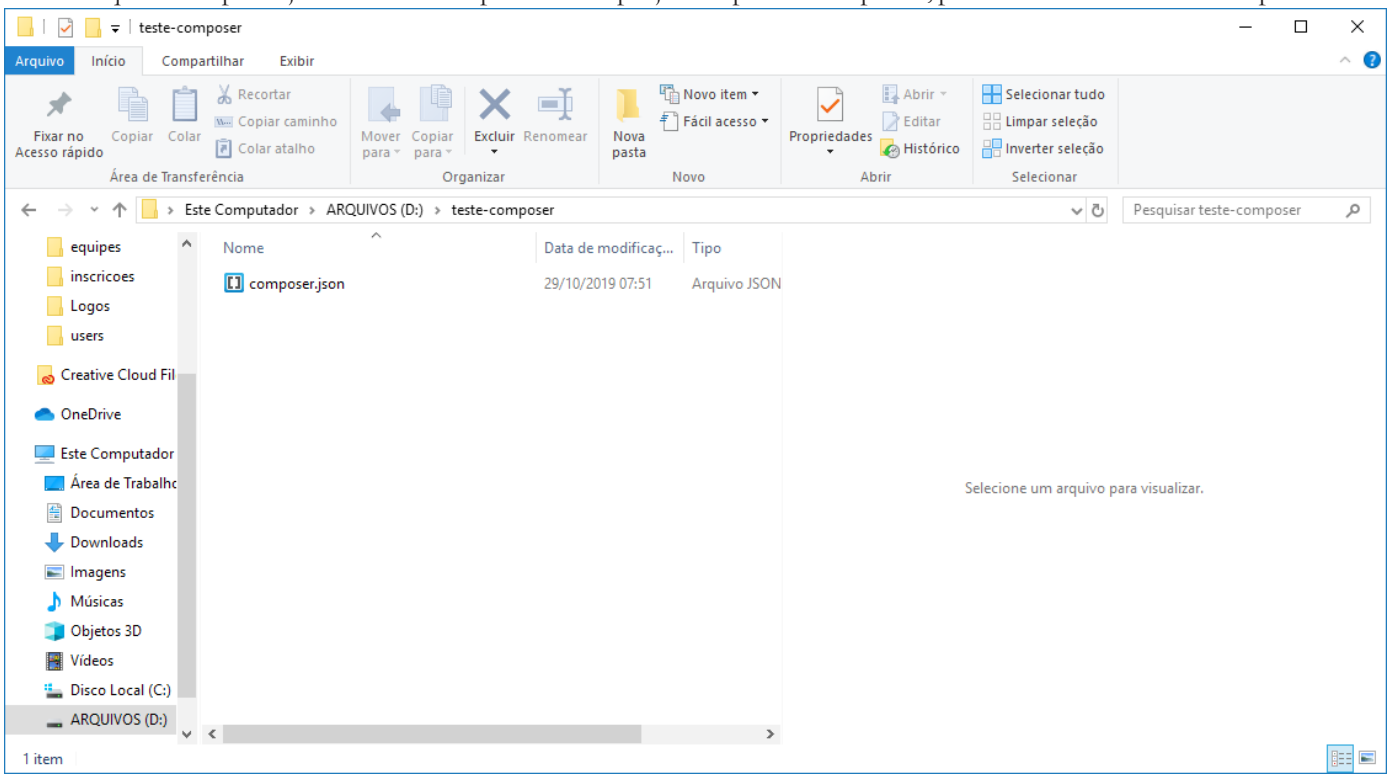
- Would you like to define your dependencies (require) interactively [yes]?
- Would you like to define your dev dependencies (require-dev) interactively [yes]?

```
Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? no
Would you like to define your dev dependencies (require-dev) interactively [yes]? no
```

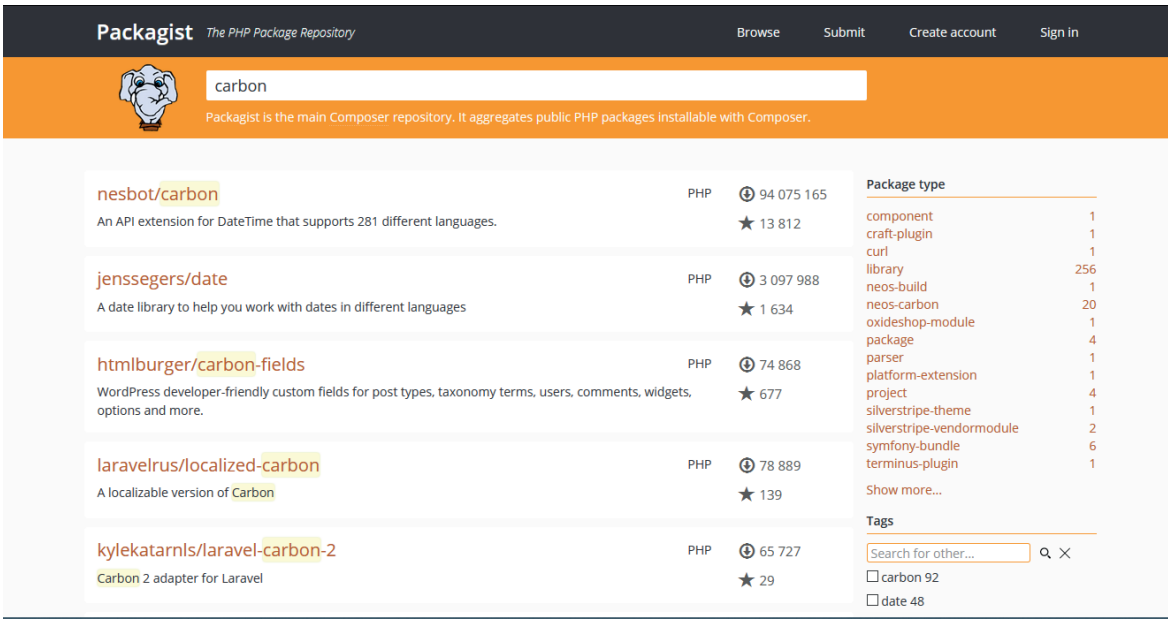
No final, o assistente te mostra como está o arquivo composer.json. Digite "yes" e tecle Enter para confirmar a geração do arquivo.

```
{
    "name": "luis/teste",
    "description": "teste",
    "type": "project",
    "require": {}
}

Do you confirm generation [yes]? yes
PS D:\teste-composer>
```

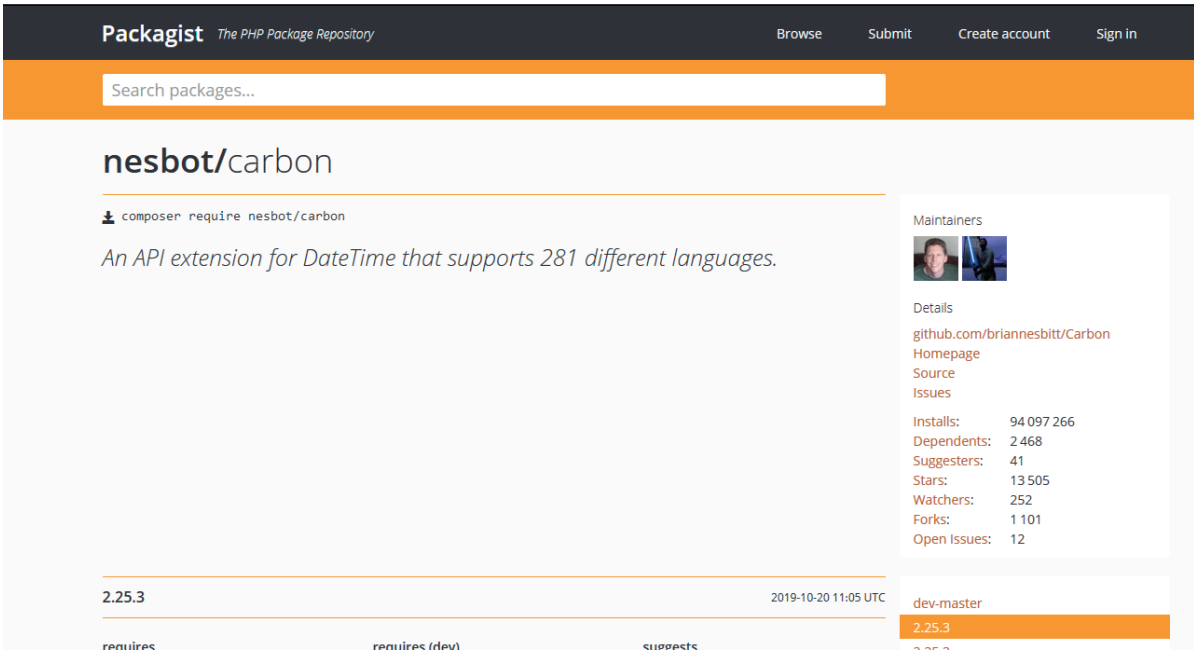
O arquivo composer.json será salvo na pasta do seu projeto. A partir desse ponto, podemos instalar as nossas dependências:



Para o nosso teste, vamos simular a instalação de um pacote muito utilizado em projetos PHP, a biblioteca de manipulação de data e hora Carbon (https://packagist.org/packages/nesbot/carbon). Para procurar a biblioteca, precisamos procurar pelo código pelo qual é identificado no packagist. Assim, na página inicial do Packagist, procure na caixa de pesquisa por "Carbon". Clique em "nesbot/carbon".



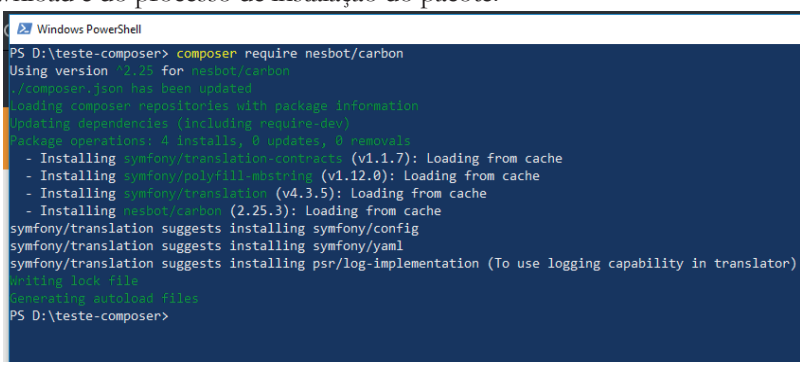
Na página do projeto no Packagist, perceba que o comando de instalação aparece. O comando sempre é **composer require**, seguido pelo código do pacote a ser instalado.



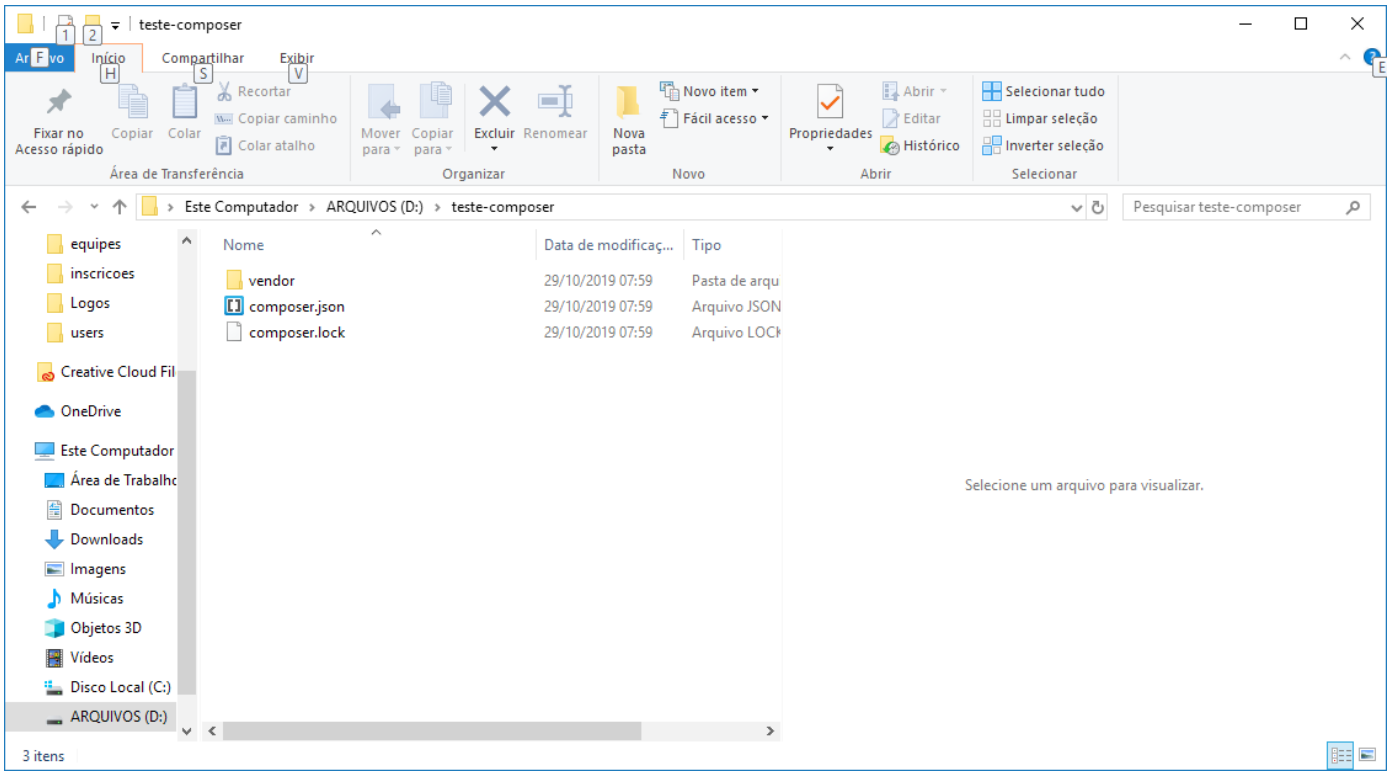
Assim, para o nosso projeto, o código da biblioteca se chama "nesbot/carbon". Assim, o comando de instalação é **composer require nesbot/carbon**. Digite esse comando no terminal da pasta do seu projeto. O Composer fará os seguintes passos:

- atualizará o arquivo composer.json;
- instalará o próprio pacote;
- instalará as dependências desse pacote.

Aguarde o fim do download e do processo de instalação do pacote.



Na pasta do projeto, aparecerá uma subpasta nova, denominada de "vendor". Lá estarão os arquivos das nossas dependências. Além de um arquivo "composer.lock", onde estarão listadas as versões cujas dependências foram instaladas.



Para usar o pacote que acabamos de instalar, precisamos fazê-lo em um arquivo PHP:

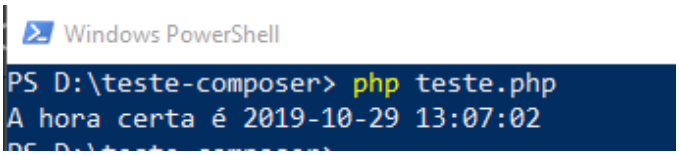
- Incluir o arquivo de autoload do Composer, para que o Composer carregue todas as dependências. Esse arquivo se chama "autoload.php" e fica dentro da pasta vendor.
- Importar as classes da nossa dependência e usar os métodos desse pacote. Isso depende de cada pacote. Consulte a documentação para saber como fazer isso.

> Na página do projeto dentro do Packagist ou no Github, é sempre comum achar as instruções de instalação, configuração e uso ou no mínimo, um link apontando para uma página com esses conteúdos. Vale a pena conferir.

Vamos, agora, criar um arquivo PHP para testar a nossa dependência. Ela terá três linhas: uma para carregar o arquivo autoload, outra para importar a classe Carbon e outra para exibir a data e hora atual, com ajuda de um método estático da classe Carbon. Esse arquivo se chama teste.php e ficará na raiz da nossa pasta de testes. Vamos ver?

```
<?php

require "vendor/autoload.php";
use Carbon\Carbon;
printf("A hora certa é %s", Carbon::now()->toDateTimeString());
```

Vamos executar esse arquivo, rodando dentro do nosso terminal da pasta do projeto o comando `php teste.php`. Se tudo der certo, a hora e data certas aparecerão na tela.



Vamos agora para mais alguns comandos úteis do Composer:

- **composer remove**: remove um pacote da pasta de dependências do projeto;
- **composer install**: instala todas as dependências que estão listadas em um arquivo composer.json. Com isso, não precisamos salvar a pasta vendor em nosso gerenciamento de versões. Basta apenas salvar o arquivo composer.json que terá uma referência dos pacotes usados pelo nosso projeto.
- **composer dumpautoload**: recria o arquivo de autoload. Útil para casos em que são adicionados novos arquivos no sistema.

E, com isso, encerramos a nossa quinta aula. Esperamos que tenha gostado. Na próxima, falaremos sobre Docker. Até lá!

Retomando a aula

Chegamos ao final da nossa quinta aula. Vamos relembrar?

1 – Sistemas de controle de dependências

Você viu aqui que quando usamos uma biblioteca pronta para o nosso projeto de software, ela passa a ser chamada de **dependência**, pois o nosso software passa a depender da nossa biblioteca para que possa funcionar adequadamente. Ela pode ser gerenciada de maneira manual, mas pode causar vários problemas. Assim, surgiram os gerenciadores de dependências, que automatizam o processo de gerenciamento.

2 – Gerenciador de pacotes composer

Nesta seção, você foi introduzido ao Gerenciador de Pacotes Composer. Vimos como funciona o processo de instalação e de uso básico da ferramenta, quando instalamos uma biblioteca para a manipulação de Data e Hora.

Vale a pena

Vale a pena **ler**

ENGHOLM JR., Hélio. *Engenharia de software na prática.* São Paulo: Novatec, 2010.

SOMMERVILLE, Ian. *Engenharia de Software.* 9. ed. São Paulo: Pearson, 2011.

Vale a pena **acessar**

BERALDO, Roberto. Composer: Agilidade no Gerenciamento de Dependências com PHP. Profissionais TI, 2015. Disponível em: https://www.profissionaisti.com.br/2015/06/composer-agilidade-no-gerenciamento-de-dependencias-com-php/. Acesso em 29 out. 2019.

COMPOSER. Disponível em: https://getcomposer.org/. Acesso em 29 out. 2019.

COMPOSER. Introduction. Disponível em: https://getcomposer.org/doc/00-intro.md. Acesso em 29 out. 2019.

OMBREDANNE, Philippe. Software Dependencies: A not-too-technical introduction. NEXB, 2016. Disponível em: https://www.nexb.com/blog/introduction_to_software_dependencies.html. Acesso em 29 out. 2019.

PACKAGIST. Disponível em: https://packagist.org/. Acesso em 29 out. 2019.

PACKAGIST. Statistics. Disponível em: https://packagist.org/statistics/. Acesso em 29 out. 2019.

VIEIRA, Rafael Nunes. Gerenciadores de dependências. Medium, 2016. Disponível em: https://medium.com/@nvieirarafael/gerenciadores-de-depend%C3%AAncias-bed99b01e94a. Acesso em 29 out. 2019

Minhas anotações