آموزش استفاده از Composer در PHP

مقدمه

چرخ را دوباره اختراع نکنید! همه ما توسعه دهنده های نرم افزار در پروژه هایی که توسعه می دهیم به ابزاری نیاز داریم که توسط دیگر توسعه دهنده ها از پیش ساخته و منتشر شده و ما می توانیم به جای بازطراحی آنها فقط تصمیم بگیریم که از آنها استفاده کنیم. اما استفاده از این ابزار ها یا به اصطلاح پکیج[1] ها خودش به پیچیدگی هایی ختم می شود که ما مجبور می شویم به دنبال راهکار هایی برای آن بگردیم که البته راه حل مناسب استفاده از یک ابزار مدیریت وابستگی است و مجددا باید گفت چرخ را دوباره اختراع نکنید!

مدیریت وابستگی[2]

همانطور که پیشتر اشاره کردیم وقتی برای توسعه یک پروژه از پکیج های آماده که توسط دیگران ساخته شده استفاده می کنیم پیچیدگی هایی پیش می آید که نیازمند مدیریت هستند. در ادامه دو مورد از این پیچیدگی ها را بررسی می کنیم.

بروزرسانی پکیج ها

پکیج ها همانند نرم افزار ها در طول زمان توسعه داده می شوند و یا باگ های موجود در آنها رفع می شود و نسخه های جدیدی از آنها منتشر می شود. به عنوان مصرف کننده باید آنها را به روز نگه داشته تا کیفیت و امنیت محصول خود را تضمین کنیم. علاوه بر به بروزرسانی مداوم پکیج های مورد استفاده باید به هنگام بروزرسانی به نسخه جدید سازگاری آن نسخه با پروژه را بررسی کنیم تا مبادا محصول ما دچار مشکل شود.

وابستگی های تو در تو

پکیج های مورد نیاز پروژه ما (وابستگی های سطح یک) برای ارائه کاربرد مورد نظر ممکن است از دیگر پکیج ها استفاده کنند و اصطلاحا خود وابستگی هایی که آنرا وابستگی های سطح دو می نامیم داشته باشند، این داستان ممکن است حتی ادامه دار باشد و وابستگی های سطح سوم، چهارم و ... هم به وجود بیایند. برای مدیریت این مسئله دو راهکار را می توان درنظر گرفت: ۱) برای هر پکیج وابستگی های آن را در دایرکتوری همان پکیج قرار بدهیم و سلسله مراتبی از وابستگی ها به وجود بیاوریم. ۲) همه پکیج ها از همه سطوح وابستگی بدون اینکه تفاوتی برای سطح آنها قائل شویم را در کنار هم قرار دهیم.
در راهکار اول مدیریت بسیار ساده تر می باشد اما با توجه به اینکه در دنیای واقعی بسیاری از وابستگی ها مشترک هستند حجم بسیار بسیار زیادی پکیج تکراری خواهیم داشت عملا این راهکار ناکارآمد است.
در راهکار دوم که معمولا راهکار برگزیده ابزار های مدیریت وابستگی می باشد، همه وابستگی ها از سطوح مختلف در کنار هم نگهداری می شوند و از دانلود مجدد یا نگهداری یک پکیج در مکان های مختلف جلوگیری کنیم. در این راهکار باید نسخه پکیج مورد نظر را با دقت انتخاب کنیم که سازگار با تمام پکیج های دیگر و پروژه اصلی باشد.

ابزار مدیریت وابستگی

با توجه توضیحاتی که پیشتر شرح دادیم لزوم وجود یک ابزار نیرومند برای مدیریت وابستگی ها در پروژه های نرم افزاری کاملا احساس می شود. مهمترین وظیفه این ابزار این است که با توجه به وابستگی های پروژه در سطوح مختلف، نسخه مناسب از پکیج های مورد نیاز را از مخزن[3] های تعریف شده دانلود کرده و در اختیار توسعه دهنده و پروژه قرار می دهند.

---

[1] Package

[2] Dependency Management

[3] Repository

برای زبان های مختلف ابزار های مختلفی طراحی شده است، برای مثال Maven و Gradle برای جاوا، pip برای Python، SBT برای اسکالا و Dep برای Go را می توان نام برد. برای زبان PHP در گذشته از ابزار Pear استفاده می شد اما سالهاست که Composer نه به عنوان ابزار رسمی اما به عنوان محبوب ترین ابزار مدیریت وابستگی در پروژه های PHP استفاده می شود.

قرارداد نامگذاری نسخه[4]

یکی از مباحث مهمی که در استفاده و توسعه پکیج ها باید به آن توجه کنیم قرارداد نامگذاری نسخه است. قرارداد های مختلفی در این زمینه وجود دارد. محبوب ترین این قرارداد ها که توسط سایت semver.org تعریف شده به این صورت است:

MAJOR.MINOR.PATCH

در این قرارداد نام نسخه پکیج (یا نرم افزار) از سه بخش تشکیل می شود که ادامه هر کدام را توضیح می دهیم.

بخش MAJOR در نسخه

بخش اول در نام نسخه Major نام دارد که نسخه اصلی پکیج به حساب می آید و تغییر آن به معنی تغییرات بنیادی یا مهم در پکیج است و لزوما با نسخه قبلی سازگار نمی باشد. در صورتی که ما توسعه دهنده پکیج هستیم و در نسخه جدید تغییراتی در API پکیج ایجاد کرده ایم که کاربران نسخه قبلی بدون تغییر نمی توانند از این نسخه استفاده کنند ما این بخش از نسخه را یک واحد افزایش می دهیم. برای مثال اگر نسخه قبلی پکیج ما v2.6.1 بوده نسخه جدید را باید v3.0.0 نامگذاری کنیم. در صورتی که ما کاربر یک پکیج هستیم باید به هنگام بروزرسانی پکیج توجه کنیم که با بروزرسانی به نسخه Major جدید ممکن است به تغییراتی در پروژه برای سازگاری با نسخه جدید نیازمند باشیم و حتی ممکن است از بروزرسانی منصرف شویم.

بخش MINOR در نسخه

بخش دوم در نام نسخه Minor نام دارد که نسخه فرعی پکیج به حساب می آید و تغییر آن به معنی معرفی شدن فیچر جدید یا بهبود قابل توجه با حفظ کامل سازگاری با نسخه قبلی است. برای مثال اگر نسخه پکیج v2.6.1 است چنین تغییری می تواند به نسخه v2.7.0 منجر شود. با توجه توضیحات ذکر شده می توان با خاطری نسبتا آسوده پکیج ها را به نسخه های فرعی جدید بروزرسانی کنیم.

بخش PATCH در نسخه

بخش سوم نام نسخه Patch نام دارد که تغییر آن معمولا بخاطر تغییرات کوچک بخصوص رفع باگ است و معمولا هیچ قابلیت جدیدی برای معرفی وجود ندارد. Patch های جدید با نسخه قبلی باید کاملا سازگار باشند.

ابزار Composer

همانطور که اشاره کردیم Composer محبوب ترین ابزار مدیریت وابستگی برای پروژه های به زبان PHP می باشد که فریم ورک های معروف PHP از جمله Laravel، Symfony و Zend Framework همگی از این ابزار استفاده می کنند. این ابزار یکی از قویترین ابزار های مدیریت وابستگی می باشد که امکانات بسیاری را در اختیار توسعه دهنده ها قرار می دهد و ما سعی کردیم در این مقاله تعدادی از آنها را معرفی کنیم.

Composer بطور پیشفرض از مخزن packagist.org برای دانلود پکیج ها استفاده می کند و می توانید پکیج های قابل استفاده را در این سایت ببینید.

نصب Composer

Composer را به عنوان یک ابزار Command Line می توانید از سایت getcomposer.org دانلود و نصب کنید. پس از نصب می توانید صحت نصب را با اجرای دستور زیر در خط فرمان (ترمینال) بررسی کنید.

```
composer --version
```

و برای افزودن پکیج های مورد نیاز به پروژه می توانید دستوری مشابه دستور زیر را در دایکتوری پروژه اجرا کنید.

---

[4] Version naming convention

```
composer require miladrahimi/phpconfig
```

در صورتی که مشکلی رخ ندهد، پس اجرای دستور بالا Composer پکیج مورد نظر را دانلود در دایرکتوری /your-project vendor قرار می دهد و همچنین فایل composer.json و composer.lock را هم در دایرکتوری پروژه ایجاد می کند. کاربرد این فایل را در ادامه توضیح می دهیم.

فایل composer.json

فایل composer.json پس از اجرای دستور init یا تعریف اولین وابستگی برای پروژه در دایرکتوری ریشه (root) پروژه ایجاد می شود. این فایل را که می توان به طور دستی و با یک Text Editor هم ساخت شامل لیست وابستگی ها (پکیج ها مورد نیاز پروژه)، مخزن ها، Auto-loader ها و همه اطلاعاتی است که Composer نیاز دارد تا درباره پروژه شما بداند می باشد.
در این فایل نسخه پکیج ها را می توان بصورت انعطاف پذیر همانند v3.* یا v3.6.* نوشت تا بروزرسانی پکیج ها و همچنین جلوگیری از نصب نسخه غیر قابل سازگار ممکن باشد.

فایل composer.lock

فایل composer.lock که در کنار فایل composer.json قرار می گیرد شامل اطلاعات دقیق تر (نسخه دقیق پکیج ها) می باشد. این فایل به هنگام اجرای دستور install مورد استفاده قرار می گیرد تا از بروزرسانی ناخواسته وابستگی ها در محیط های حساس جلوگیری شود. پیشنهاد می شود این فایل به همراه پروژه به سرور منتقل شود تا با نصب وابستگی ها در سرور مطمئن باشیم همان نسخه استفاده شده در محیط تست و یا توسعه بر بروی سرور نصب می شود.

دایرکتوری vendor

این دایرکتوری شامل تمام پکیج های دانلود شده توسط Composer می باشد که مورد نیاز پروژه است. علاوه بر پکیج های نصب شده، اطلاعات مربوط به نحوه بارگذاری آنها و همچنین فایل autoload.php در این دایرکتوری نگهداری می شود.
با توجه به اینکه با استفاده از اطلاعات موجود در فایل های composer.json و composer.lock و نرم افزار Composer هر زمان و هر کجا می توان مجددا تمام پکیج های مورد نیاز پروژه را نصب کرد به هنگام به اشتراک گذاری، آپلود به سرور یا مخزن Git و Deploy این دایرکتوری را ignore کرد.

تعریف وابستگی ها

در صورتی که در پروژه خود به یک پکیج نیاز دارید ابتدا باید آنرا در مخزن packagist.org پیدا کنید. در صفحه مربوط به پکیج در سایت Packagist می توانید روند افزودن آن به پروژه مشاهده کنید. برای مثال برای افزودن پکیج PhpRouter به پروژه می توانید دستور زیر را در دایرکتوری پروژه اجرا کنید.

```
composer require miladrahimi/phprouter
```

با افزودن اولین وابستگی به پروژه، Composer دو فایل composer.lock و composer.json و همچنین دایرکتوری vendor را به پروژه شما اضافه می کند و در صورت وجود این فایل ها و دایرکتوری بروزرسانی می شود.
پس از افزودن وابستگی به پروژه، Composer آنرا دانلود کرده و در دایرکتوری vendor قرار می دهد و همچنین وابستگی مورد نظر را به بخش require در فایل composer.json اضافه می کند و نهایتا composer.lock را هم بروزرسانی می کند.
در صورتی که پکیج مورد نظر تنها کاربرد تستی دارد و در سرور اصلی نیازی به نصب آن نیست می توانید با دستوری مشابه دستور زیر آنرا به پروژه اضافه کنید.

```
composer require phpunit/phpunit --dev
```

دستور بالا پکیج PhpUnit را به بخش require-dev در composer.json اضافه خواهد کرد.
چنانچه نسخه خاصی از پکیج مد نظرتان هست می توانید همانند دستور زیر آنرا به پروژه به اضافه کنید:

```
composer require miladrahimi/phprouter:"3.0"
```

با اجرای دستور بالا نسخه v3.0.* از پکیج را نصب می شود. به مثال دیگری که در ادامه آورده ایم هم توجه کنید.
composer require miladrahimi/phprouter:"3.*"
با اجرای این دستور نسخه v3.* از پکیج نصب خواهد شد. Composer جدیدترین نسخه از پکیج که بخش Major آن 3 می باشد را نصب خواهد کرد.

برای مشاهده دیگر فرمت های نوشتن نسخه که توسط Composer معرفی شده است می تواند به مقاله رسمی سایت Composer با نام Versions and constraints مراجعه کنید.

بارگذاری پکیج ها

همانطور که قبلا گفتیم Composer پکیج های مورد نیاز پروژه (و دیگر پکیج ها) را پس از دانلود در دایرکتوری vendor قرار می دهد. Composer همچنین فایلی به نام autoload.php در این دایرکتوری ایجاد میکند. این فایل حاوی Auto-loader برای تمام پکیج های نصب شده توسط Composer است و با include کردن آن در پروژه می توان از پکیج های نصب شده استفاده کرد. مثال زیر نحوه استفاده از پکیج PhpRouter را نشان می دهد:

```
<?php

include "vendor/autoload.php";

use MiladRahimi\PhpRouter\Router;

$router = new Router();

$router->get('/', function () {
    return 'This is home page!';
});

$router->dispatch();
```

بروزرسانی وابستگی ها

پس از نصب وابستگی ها ممکن است نیاز باشد که آنها را بروزرسانی کنیم که در آن صورت می توانید از دستوری مشابه دستور زیر استفاده کنید.

```
composer update miladrahimi/phprouter
```

پس از اجرای دستور بالا جدیدترین نسخه سازگار از پکیج مورد نظر توسط Composer دانلود و جایگزین نسخه قبلی آن در دایرکتوری vendor می شود. همچنین فایل composer.lock هم بروزرسانی می شود اما هیچ تغییر در فایل composer.json نخواهیم داشت.

در صورتی که می خواهید همه وابستگی ها را بروزرسانی کنید می توانید از دستور زیر استفاده کنید.

```
composer update
```

این دستور تمام وابستگی ها، دایرکتوری vendor و فایل composer.lock را بروزرسانی می کند. این دستور را با دستور زیر که خود نرم افزار Composer را بروزرسانی می کند اشتباه نگیرید.

```
composer self-update
```

نصب وابستگی ها

همانطور که قبلا توضیح دادیم بهتر است دایرکتوری vendor را به هنگام آپلود پروژه در سرور یا در مخزن هایی مانند گیتهاب ignore کرد. همچنین در صورتی که بطور کلاسیک برای Deploy پروژه روی سرور آنرا آپلود می کنید در این صورت به هنگام دریافت پروژه از مخزن دایرکتوری vendor را نداریم اما می توان با دستور زیر مجددا وابستگی ها را نصب کنیم.

```
composer install
```

دستور بالا در صورتی که فایل composer.lock در دسترس باشد، پکیج ها را مطابق با نسخه تعریف شده در آن نصب می کند و در غیر این صورت با استفاده از اطلاعات موجود در فایل composer.json آخرین نسخه سازگار با پروژه را نصب خواهد کرد.

در محیط هایی همانند سرور اصلی با استفاده از پارامتر no-dev— می توانید از نصب پکیج های مورد نیاز محیط های تستی جلو گیری کنید.

```
composer install --no-dev
```

Autoload

یکی از قابلیت های مفیدی که Composer ارائه می دهد که البته ممکن است فراتر از وظایف یک ابزار مدیریت وابستگی باشد، امکان تعریف autoload برای پروژه است. در بخش های قبل متوجه شدیم که برای بارگذاری پکیج های دانلود شده توسط Composer باید فایل vendor/autoload.php در پروژه خود include کنید. از طرفی پروژه شما هم به یک Auto-loader برای بارگذاری کلاس های پروژه نیاز دارد. با قابلیت مورد نظر نیاز نیست که شما به طور جداگانه یک Auto-loder طراحی کنید تنها کافیت تا اطلاعات بارگذاری پروژه را به composer.json اضافه کنید تا همان autoload موجود در دایرکتوری vendor کلاس های پروژه شما را هم بارگذاری کند.

```
"autoload": {
    "psr-4": {"App\\": "app/"}
}
```

مخزن ها

Packagist مخزن رسمی و عمومی Composer است و Composer برای نصب پکیج ها به این مخزن رجوع می کند. با وجود مخزن Packagist برای پکیج های خصوصی و یا پکیج هایی که در این مخزن رجیستر نشده اند ما به مخزن های ثانویه نیازمندیم که خوشبختانه Composer به راحتی به ما اجازه می دهد تا این مخزن ها را برای پروژه تعریف کنیم.
در صورتی که مخزن مورد یک مخزن خصوصی در یک سرور Git همانند GitHub یا GitLab است می توانید با آنرا همانند کد زیر به فایل composer.json اضافه کنید.

```
"repositories": [
    {
        "type": "git",
        "url": "https://github.com/private-company/foo"
    }
],
"require": {
    "private-company/foo": "1.*"
}
```

در صورتی که پکیج مورد نظر شما فقط در قالب تعدادی فایل در دسترس است می توانید به شکل زیر آنرا در composer.json اضافه کنید.

```
"repositories": [
    {
        "type": "path",
        "url": "../../packages/my-package"
    }
],
"require": {
    "my/package": "*"
}
```

برای اطلاعات بیشتر در زمینه مخزن ها می توانید بخش Repositories از مستندات رسمی Composer را مطالعه کنید.

گفتار پایانی

در این مقاله با مفاهیم مقدماتی از Composer آشنا شدید اما برای مطالعه بیشتر و آشنایی بیشتر با قابلیت های این ابزار مفید می توانید به مستندات رسمی مراجعه کنید.