

# 11. Контейнер зависимостей. Блог на PHP

Червень 28, 2021

В этом уроке, мы добавим контейнер зависимостей, который поможет создавать PHP объекты централизованно. Для этого, мы воспользуемся библиотекой `php-di/php-di`, которую можно установить с помощью `composer`.

## Какие темы мы разберем

В [11. Контейнер зависимостей](#) уроке мы рассмотрим следующие темы:

- Что такое Контейнер зависимости и внедрение зависимостей (Dependency Injection)
- Как добавить поддержку Контейнера зависимостей в проект
- Как создать и загрузить необходимую конфигурацию классов для создания объектов
- [Бонус] Добавим поддержку встроенного PHP сервера

Приятного просмотра:

## 11. Контейнер зависимостей. Блог на PHP



В рамках урока, мы напишем следующий код.

Начнем с самого главного, а именно, добавим возможность “поднимать” блог на встроенном PHP сервере. Для ссылки, вы можете запустить PHP сервер без дополнительного сервера с помощью команды:

```
php -S localhost:9000
```

В ответ, мы увидим сообщение об успешном запуске сервера. И можно с уверенностью переходить по ссылке с терминала.

```
[Mon Jun 28 06:41:20 2021] PHP 7.4.20 Development Server (http://local
```



В классе `AssetExtension`, давайте добавим проверку на существование ключа `REQUEST_SCHEME`. Так как в случае с девелоперским сервером (смотрите выше), такой переменной может не быть, и все ссылки на картинки могут сломаться.

```
public function getBaseUrl(): string
{
    $params = $this->request->getServerParams();
    $scheme = $params['REQUEST_SCHEME'] ?? 'http';
}
```

```
return $scheme . '://' . $params['HTTP_HOST'] . '/';  
}
```

Далее, добавим новую библиотеку `php-di/php-di` через `composer`. Для этого, запустим команду:

```
composer require php-di/php-di:6
```

После успешной установки, создадим новый файл `config/di.php`. Файл `di.php` будет содержать настройки и зависимости, для создания объектов. В этом уроке, мы перенесем создание двух классов из `index.php` файла и добавим конфигурации в `di.php`.

Новый файл `config/di.php` выглядит следующим образом:

```
<?php  
  
declare(strict_types=1);  
  
use Twig\Environment;  
use Twig\Loader\FilesystemLoader;  
use function DI\autowire;  
use function DI\get;  
  
return [  
    FilesystemLoader::class => autowire()  
        ->constructorParameter('paths', 'templates'),  
  
    Environment::class => autowire()  
        ->constructorParameter(  
            'loader',  
            get(FilesystemLoader::class)  
        )  
];
```

В `index.php` файле, давайте создадим объект класса `ContainerBuilder` и вызовем метод `addDefinitions` с указанием файла конфигурации `di.php`. Создание

объекта Контейнера происходит в момент вызова метода `$builder->build()`. Также, для того, что бы Slim библиотека могла пользоваться новым Контейнером, мы добавляем его с помощью метода `setContainer`.

```
use DI\ContainerBuilder;

//код сразу после require __DIR__ . '/vendor/autoload.php';
$builder = new ContainerBuilder();
$builder->addDefinitions('config/di.php');

$container = $builder->build();

AppFactory::setContainer($container);
```

С помощью Контейнера и метода `get()`, давайте вернем объект `Twig\Environment` и присвоим в переменную `$view`.

Файл `index.php`.

```
$view = $container->get(Environment::class);
$app->add(new TwigMiddleware($view));
```

Скачать код данного урока можно по [ссылке](#).

[Підписуйтеся на канал “Спільнота програмістів – Developer & Code” в телеграмі](#)



**Макс Пронько** – Програміст, CEO компанії Pronko Consulting, розбираємо Веб технології. Автор YouTube каналу [Макс Пронько](#). [Телеграм](#) група сайту.

[← 10. Пагинация постов. Блог на PHP](#)

[12. Адаптер для подключения к базе данных. Блог на PHP →](#)

2024 © Developer & Code. Усі права захищені. Зроблено з ❤️ для UA.