

MASARYK UNIVERSITY
FACULTY OF INFORMATICS



Refactoring of Kdyby packages

BACHELOR'S THESIS

Filip Procházka

Brno, Spring 2017

MASARYK UNIVERSITY
FACULTY OF INFORMATICS



Refactoring of Kdyby packages

BACHELOR'S THESIS

Filip Procházka

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Filip Procházka

Advisor: RNDr. Jaroslav Bayer

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

package, kdyby, nette, doctrine, orm, composer, packagist

Contents

1	Introduction	1
2	Background for understanding Kdyby	2
2.1	<i>A brief history of Kdyby</i>	2
2.2	<i>Techniques and design patterns</i>	2
2.3	<i>Technologies used</i>	3
3	Current state of Kdyby	8
3.1	<i>State of the project</i>	8
3.2	<i>State of each package</i>	9
4	Designing roadmap of refactoring	26
4.1	<i>Deprecations</i>	26
4.2	<i>Common requirements</i>	26
4.3	<i>Specific requirements for each package</i>	28
5	The refactoring process of Kdyby	32
5.1	<i>Doctrine</i>	32
5.2	<i>Console</i>	32
5.3	<i>Events</i>	32
5.4	<i>Annotations</i>	32
5.5	<i>DoctrineCache</i>	32
5.6	<i>DoctrineMagicAccessors</i>	32
5.7	<i>DoctrineCollectionsReadonly</i>	32
5.8	<i>DoctrineCollectionsLazy</i>	33
5.9	<i>DoctrineDbalBatchImport</i>	33
5.10	<i>DoctrineForms</i>	33
5.11	<i>Autowired</i>	33
5.12	<i>FormsReplicator</i>	33
5.13	<i>Translation</i>	33
5.14	<i>Validator</i>	33
5.15	<i>RabbitMq</i>	33
5.16	<i>Money</i>	33
5.17	<i>DoctrineMoney</i>	34
5.18	<i>Aop</i>	34
5.19	<i>Clock</i>	34

5.20	<i>Redis</i>	34
5.21	<i>ParseUseStatements</i>	34
5.22	<i>RedisActiveLock</i>	34
5.23	<i>TesterParallelStress</i>	34
5.24	<i>Monolog</i>	34
5.25	<i>ElasticSearch</i>	34
5.26	<i>DoctrineSearch</i>	35
5.27	<i>Geocoder</i>	35
5.28	<i>CsobPaygateNette</i>	35
5.29	<i>CsobPaymentGateway</i>	35
5.30	<i>Wkhtmltopdf</i>	35
5.31	<i>FakeSession</i>	35
5.32	<i>RequestStack</i>	35
5.33	<i>StrictObjects</i>	35
5.34	<i>Facebook</i>	35
5.35	<i>Google</i>	36
5.36	<i>Github</i>	36
5.37	<i>NettePhpServer</i>	36
5.38	<i>TesterExtras</i>	36
5.39	<i>HtmlValidatorPanel</i>	36
6	Conclusion and future work	37

List of Tables

List of Figures

- 3.1 Supported PHP Versions. Green is active support, orange is security fixes only. Up-to-date version is at <http://php.net/supported-versions.php> 9
- 3.2 Example of PHPDoc with annotations, that the doctrine/annotations can parse from source code. 11
- 3.3 Example of working with doctrine/collections. 13
- 3.4 Example of DoctrineCollectionsReadOnly in entity. 14

1 Introduction

Kdyby is an Open-source software (OSS) [1] project that I, Filip Procházka, lead and maintain. It is a set of PHP [2] libraries, that aim to ease writing of web applications.

Through my carer, I have used Kdyby in core business applications of companies such as Damejidlo.cz and Rohlik.cz. A lot of people consider my work useful enough to incorporate it to their own applications as well.

As of writing this, the more popular libraries have hundreds of thousands of downloads. Five of Kdyby libraries have over quarter million downloads and one is approaching half a million with staggering amount of 470 thousands of downloads [3]. In conclusion, a sober estimate would be, that Kdyby libraries are used in hundreds of real production applications.

If I account only for the two biggest projects that I can confirm are using Kdyby packages, over a billion Czech crowns¹ has literary flowed through Kdyby. That is a big responsibility.

Over the years, I have had problems keeping up with the demand and the packages began to get obsolete. I wanna use this thesis as way to fix the situation.

I will review the state of each library and decide its future. Which means I will either deprecate it and provide the users a suggestion for a better alternative, or fix the problems and refactor the library.

1. Rohlik.cz loni prodal zboží za miliardu, letos chce konečně zisk
<http://tyinternety.cz/e-commerce/rohlik-cz-loni-dosahl-na-miliardovy-obrat-letos-chce-konecne-zisk/>

2 Background for understanding Kdyby

2.1 A brief history of Kdyby

In 2006 I have started working on my own Content management system (CMS) [4]. A prototype was used in production on few websites I created. The oldest preserved version is [archived on my Github](#)¹. It is a great learning material on how to not write a CMS.

Then the concept of OSS [1] was introduced to me and I have decided to start working on everything openly, under a free license [5]. Sadly, since then, no new working version of Kdyby CMS was ever released, because I have rewritten it from scratch exactly 10 times.

In 2012, I have decomposed the the emerging system into separate libraries that can be used more or less independently and have their own release cycle. This approach was preserved to this day.

2.2 Techniques and design patterns

2.2.1 Dependency Injection

Inversion of control is a design principle in which custom-written portions of a computer program receive the flow of control from a generic framework.

Dependency Injection (DI) is a technique whereby one object supplies the dependencies of another object. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern [6].

2.2.2 Aspect Oriented Programming

In computing, Aspect-oriented Programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a point-cut specification, such as log all function calls when the name of the

1. archived on my Github <https://github.com/fprochazka/kdyby-cms-old>

function begins with 'set'. This allows behaviors that are not central to the business logic (such as logging) to be added to a program without cluttering the code core to the functionality [7].

2.2.3 Event Dispatcher

The Event Dispatcher is a pattern for writing modular code. It allows to create extension points in the library or application that another library or application can hook into and change or extend the behavior.

Typically, the extension points are called hooks or events and the new functionality is provided with objects called listeners.

2.3 Technologies used

2.3.1 Git and Github

Git is a Version Control System, that is decentralized and considered very fast [8]. Github is a collaboration platform for software development using Git.

Each project has a page on Github called a repository, that can be used to inspect the Git history, files and other metadata. On the project repository page, there are issues and pull requests. Pull requests are a way to ask the maintainer of the repository to incorporate provided code patch to the repository. It can be a bugfix or feature.

There are tools around pull requests that allow collaboration, a code review and discussion about the provided code, so that the maintainers can help the contributors to provide the best code possible.

Kdyby is hosted and developed on Github, with the help of several other maintainers and the community, that contributes bugfixes and features.

2.3.2 Continuous Integration

Continuous Integration (CI) is a practice of merging all developer working copies to a shared mainline several times a day, to prevent merging conflicts. [9] But now-days, the term has established to mean CI servers that run prepared task on the provided code.

In practice, it means that as the developer is working on a feature or bugfix, they push the work in progress code into a repository, the code is then picked up by a CI server that executes the tests, checks coding style and runs various other tasks to verify that the code was not broken.

When the work is finished and all the task on CI server completed with success, the code can be probably safely integrated, providing that the tests for new or changed functionality were added.

Some popular CI services are [Travis CI](https://travis-ci.org/)², [CircleCI](https://circleci.com/)³ and [GitLab CI](https://about.gitlab.com/features/gitlab-ci-cd/)⁴. Kdyby is using the Travis CI, that is free for OSS projects.

2.3.3 Nette Framework

Nette Framework (Nette) is an OSS framework for creating web applications in PHP [10]. Nette is separated into many packages.

The Dependency Injection component [nette/di](https://github.com/nette/di)⁵ provides a Dependency Injection Container (DIC) that holds the services. The component also allows to prepare a preconfigured DIC, which is then compiled into a PHP class that contains optimized code for the service creation. This compiled DI Container class is cached for reuse. There is a concept (and a class) `CompilerExtension` that allows the developer to hook into the process of configuring and compiling of the DIC. All of Kdyby packages that are an integration of some other library or tool into Nette provide a `CompilerExtension` to make the installation easy.

2.3.4 Composer

Composer is a tool for dependency management [11] in PHP. It allows you to declare the libraries your project depends on and it will manage (install or update) them for you [12].

Packages are usually published using Github with metadata in a file named `composer.json`, that is written in JSON [13] format.

2. Travis CI <https://travis-ci.org/>

3. CircleCI <https://circleci.com/>

4. GitLab CI <https://about.gitlab.com/features/gitlab-ci-cd/>

5. `nette/di` <https://github.com/nette/di>

Composer is decentralized, but has a single main metadata repository [Packagist](https://packagist.org/)⁶. It stores and provides all the package metadata like available versions and where to download them.

All Kdyby libraries are published as Composer packages on Packagist and installing them using the Composer is the only officially supported installation method.

2.3.5 OAuth 2

OAuth is a protocol for authentication and authorization that can be implemented into a web service. It is designed for secure exchange of user information, allowing third party websites to implement a login and registration process that simplifies these tasks for the user essentially allowing them to login or register to services through the OAuth 2 provider with two clicks.

Kdyby provides packages for integrating Nette with OAuth 2 providers, such as [Facebook](#)⁷, [Google](#)⁸ and [Github](#)⁹.

2.3.6 dibi

Dibi is a Database Abstraction Library for PHP. It supports a lot of significant databases: MySQL, PostgreSQL, SQLite, MS SQL, Oracle, Access and generic PDO and ODBC [14].

2.3.7 Doctrine 2 ORM

Doctrine 2 ORM is an Object–Relation Mapper (ORM) [15], which means it allows the programmer to create PHP classes called entities, that represent relational data in a database and are used to actually map the data from the database to the classes and back. In conclusion, it allows the programmer to write fully Object-oriented (OOP) [16] applications.

6. Packagist <https://packagist.org/>

7. Facebook <https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow>

8. Google <https://developers.google.com/identity/protocols/OAuth2>

9. Github <https://developer.github.com/v3/oauth/>

2.3.8 Symfony Framework

Symfony Framework (Symfony) is a PHP web application framework and a set of reusable PHP components/libraries, similar to Nette [17].

For extensibility, Symfony has Bundles, that provide similar functionality, to Nette CompilerExtension, but they operate on a different level of abstractions. Bundle is a whole package, that contains the Bundle definition, DIC extension for configuring the Bundle and it may contain adapter classes. Bundles are registered in the AppKernel of Symfony application.

2.3.9 RabbitMQ

RabbitMQ is OSS message broker software (sometimes called message-oriented middleware) that implements the Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover [18].

2.3.10 ElasticSearch

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. It is the most popular enterprise search engine [19].

2.3.11 Redis

Redis is an in-memory database OSS project that is networked, in-memory, and stores keys with optional durability [20].

2.3.12 PhpStan

PHPStan focuses on finding errors in your code without actually running it. It catches whole classes of bugs even before you write tests for the code [21].

2.3.13 PHP Standards Recommendations

The Framework Interoperability Group (FIG), which is a group of representatives from established OSS projects, discusses and creates PHP Standards Recommendations (PSR). The goal is to discover commonalities in libraries that solve similar problems. The PHP ecosystem is fragmented around tens of frameworks and libraries that all do the same, but slightly differently. This is hugely caused by the absence of a good dependency management tool like Composer, which is still very young. The PSR contain interfaces that were agreed upon for the libraries to implement. The accepted standards are listed at the [FIG website](http://www.php-fig.org/psr/)¹⁰ [22].

2.3.14 Monolog

Monolog is a logging library that sends your logs to files, sockets, inboxes, databases and various web services. This library implements the PSR-3 interface that can be typehinted against in other libraries to keep a maximum of interoperability [23].

2.3.15 Nette\Tester

Nette\Tester is an unit testing [24] framework for the PHP [25].

10. FIG website <http://www.php-fig.org/psr/>

3 Current state of Kdyby

To be able to lay out the roadmap, first we have to know the current state of each Kdyby package, the original purpose and the current requirements. We will only review those packages that actually made it to production and at least one usable version was released.

I have created few GitHub repositories as a reminder for me to start working on some other web application development problems. I did start to work on some of them, for example on DoctrineForms, but it was never "officially released". We will not discuss these incomplete packages in this thesis.

3.1 State of the project

The most relevant problem is the compatibility with new versions of the libraries, that Kdyby integrates. Current version of Nette is 2.4 and the 3.0 is being developed, but some of Kdyby packages support only Nette 2.2 or older.

The other problems appear only when you interact with the source code which is still really important for me as a maintainer and for the contributors. Also good maintainable code attracts more programmers to use it and contribute. But, there is no coding standard being enforced automatically on any package and no static analysis tool is checking the code. On the other hand almost all of the packages have unit and integration tests and linter checking the code for multiple versions of PHP.

Most of the packages are compatible with PHP 5.4, but [PHP 5.4 had end of life at 2015-09-03](#)¹ and is no longer supported by PHP developers.

The PHP 5.5 had end of life at 2016-06-21 and PHP 5.6 is currently in the phase of security fixes only and will have end of life at 2018-12-31. As you can see on the graph 3.1, everyone should be migrating to PHP 7.0 by now. But the developer has to consider what PHP versions the libraries support, upgrade them first and then migrate the application.

1. PHP 5.4 had end of life at 2015-09-03 <http://php.net/eol.php>

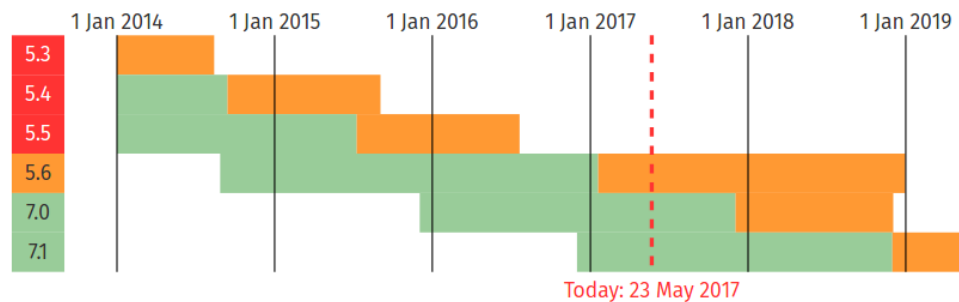


Figure 3.1: Supported PHP Versions. Green is active support, orange is security fixes only. Up-to-date version is at <http://php.net/supported-versions.php>

3.2 State of each package

This section reviews each package separately, considers the original purpose and sums up the current state.

3.2.1 Doctrine

Kdyby\Doctrine is an integration of Doctrine 2 ORM into Nette. But it has cumulated a lot of responsibilities, that don't belong to it

Doctrine 2 ORM itself is separated into several packages, mainly [doctrine/orm](https://github.com/doctrine/doctrine2)², [doctrine/common](https://github.com/doctrine/common)³, [doctrine/annotations](https://github.com/doctrine/annotations)⁴, [doctrine/cache](https://github.com/doctrine/cache)⁵ and [doctrine/collections](https://github.com/doctrine/collections)⁶. What started as a monolith integration in Kdyby, got separated into Kdyby\Events 3.2, Kdyby\Console 3.2, Kdyby\Annotations 3.2 and Kdyby\DoctrineCache 3.2 for reusability.

I have already started extracting few of them in the past, for example an entity prototyping tool 3.2, collection utilities 3.2, 3.2 and helper for loading big SQL scripts to the database 3.2.

2. doctrine/orm <https://github.com/doctrine/doctrine2>

3. doctrine/common <https://github.com/doctrine/common>

4. doctrine/annotations <https://github.com/doctrine/annotations>

5. doctrine/cache <https://github.com/doctrine/cache>

6. doctrine/collections <https://github.com/doctrine/collections>

There is a big issue [Chop up the package](#)⁷ that discusses what other parts should be separated and dropped completely.

New versions of Nette and Doctrine 2 ORM were released and completely new versions are being prepared, which the integration cannot be currently used with.

3.2.2 Console

Kdyby\Console is an integration of Symfony Framework Console Component (Symfony Console), that allows for writing interactive CLI applications. Kdyby\Doctrine depends on this package and is the reason this package exists.

There are tasks, that are better suited for console interaction, than a web interface. Among others, Doctrine 2 ORM has tools for generating a database schema from the entities metadata and there is a console command for it, that is written using Symfony Console.

The library is solving several problems with CLI applications in Nette, that are no longer relevant or have to be refactored. One of the issues is that it introduced level of abstraction above Nette Front Controller to solve generating URLs in CLI. Nette was refactored to account for this problem and this part of Kdyby\Console can be dropped altogether.

3.2.3 Events

Kdyby\Events provides an event dispatcher implementation for Nette.

It started as an integration of Doctrine 2 ORM EventManager, but then it evolved into a standalone system with support for lazy initialization of listeners and it also contains a naive bridge for Symfony Framework EventDispatcher Component (Symfony EventDispatcher).

Creating such interchangeable eventing system turned out to be a mistake. The bridging of the subscriber classes is fragile and requires a lot of compromises from the programmer. It is also a maintenance hell. The systems should have stayed separate.

7. Chop up the package <https://github.com/Kdyby/Doctrine/issues/238>

```
/**
 * @Entity
 */
class Comment
{
    /**
     * @Id
     * @GeneratedValue
     * @Column(type="string")
     */
    private $id;

    /**
     * @ManyToOne(
     *     targetEntity="User",
     *     inversedBy="comments"
     * )
     */
    private $author;
}
```

Figure 3.2: Example of PHPDoc with annotations, that the doctrine/annotations can parse from source code.

3.2.4 Annotations

Kdyby\Annotations is a simple integration of doctrine/annotations into Nette. It was created solely for the purposes of Kdyby\Doctrine, but it can be used in any Nette application that requires annotations for some functionality.

The problem the doctrine/annotations solves is that PHP doesn't have native annotations. But Doctrine simulates them using PHPDoc. The example code 3.2 illustrates usage of Doctrine 2 ORM annotations on entities. Doctrine 2 ORM can read this as metadata and also generate SQL schema for relational databases.

Kdyby\Annotations is almost up-to-date, because it is very simple and the old version for Nette still works well for the current versions.

3.2.5 DoctrineCache

Kdyby\DoctrineCache integrates doctrine/cache, that is used by doctrine/annotations and doctrine/orm to store metadata, results of various parsers and even query results.

This package has no Nette CompilerExtension, but only a helper class that configures the caching services and is used by CompilerExtensions in Kdyby\Annotations and Kdyby\Doctrine. As such, there are only few minor problems and inefficiencies that have to be fixed to work perfectly with current Nette and Doctrine.

3.2.6 DoctrineMagicAccessors

Kdyby\DoctrineMagicAccessors is a prototyping tool for writing less code in entities. It allows to not write getters and setters for entity properties.

In Doctrine 2 ORM there is an entity lazy-loading feature, that in order to work the entity cannot have any public properties, only protected or private. Which means that to be able to access the properties the developer has to define methods on the entity. This is completely correct, but when prototyping an application, it might not be obvious what methods will be required and what data they should allow to be written into the entity. This package aims to solve that, by allowing not to write the accessor methods, because it makes them available dynamically.

It was extracted from Kdyby\Doctrine and currently only exists to ease migrating away from this technique. The problem this package solves is still valid because manually writing getters and setters in PHP is tedious and error-prone, but the way the package does it creates more problems than it solves. Having the manually written accessors checked by static-analysis tool like PHPStan and unit tested is a better option.

3.2.7 DoctrineCollectionsReadonly

Entities in Doctrine 2 ORM can have associations in them. For example a cart entity might contain collection of order items. Adding an order item to the cart might be done using a `addOrderItem` method or through the collection directly as shown in 3.2.

```
class Cart
{
    private $orderItems;

    public function __construct() {
        $this->orderItems = new ArrayCollection();
    }

    public function addOrderItem(
        OrderItem $orderItem
    ): void {
        $this->orderItems->add($orderItem);
    }

    public function getOrderItems(): Collection {
        return $this->orderItems;
    }
}

$cart = new Cart()

// modifying the collection through entity API
$cart->addOrderItem(new OrderItem());

// modifying the collection outside of entity
$cart->getOrderItems()->add(new OrderItem());
```

Figure 3.3: Example of working with doctrine/collections.

When we try to modify the collection outside of the entity that owns it, we are breaking the encapsulation of that entity - it no longer has the control over what is in the collection. Therefore allowing the programmer to access the mutable collection directly is a bad practice.

The package Kdyby\DoctrineCollectionsReadOnly provides a decorator for collections, that disables methods for mutation of the collection, but keeps available those that only read data. This allows the entity to expose the collection, so that the programmer can use the friendly collections API, without having to worry to accidentally modify the internal state of the owning entity.


```
public function getOrderItems(): Collection {  
    return new ReadOnlyCollectionWrapper(  
        $this->orderItems  
    );  
}  
  
// returns first OrderItem  
$orderItem = $cart->getOrderItems()->first();  
  
// throws exception  
$cart->getOrderItems()->add(new OrderItem());
```

Figure 3.4: Example of DoctrineCollectionsReadOnly in entity.

Kdyby\DoctrineCollectionsReadOnly depends only on doctrine/collections which did not change drastically and there no changes required for the package to function with new versions.

3.2.8 DoctrineCollectionsLazy

Kdyby\DoctrineCollectionsLazy package provides implementation of doctrine/collections Collection interface, that accepts a generator function and evaluates it lazily, only when the items are actually accessed.

Kdyby\DoctrineCollectionsLazy depends only on doctrine/collections and therefore the situation is the same as with Kdyby\DoctrineCollectionsReadOnly.

3.2.9 DoctrineDbalBatchImport

Doctrine 2 ORM does not have any tool for importing a big SQL file and this package provides helpers for it. It allows iterating over huge file in a memory effective way, executing each SQL query it finds in it.

Kdyby\DoctrineDbalBatchImport has no stable version yet, but as a former part of Kdyby\Doctrine, it has to be refactored and the stable versions released.

3.2.10 Curl

PHP has a binding for the Curl library and exposes its functions to the programmer. But the API is the same as the underlying C API, which is not suited for modern Object–Oriented Programming (OOP) development. Kdyby\Curl is wrapping the Curl functions in Object–Oriented (OO) API.

There are now better and more popular packages for doing HTTP requests in PHP like [guzzlehttp/guzzle](https://github.com/guzzle/guzzle)⁸. Therefore this package is now deprecated and unmaintained.

3.2.11 CurlCaBundle

For doing secured HTTP requests over HTTPS the client must have available the public certificates, that signed the private certificates the website is using to establish the secured connections. There are hosting providers, that do not regularly update the certificates in the operating system. This creates a problem, that this package solves.

I have a cron job on my Virtual Private Server (VPS), that regularly downloads fresh certificates from Mozilla browser, extracts them to format that the Curl clients can use and then publishes them as a new version of Kdyby\CurlCaBundle.

Using this package in application means it no longer depends on outdated system certificates and they can be updated regularly with this package.

The Composer authors created their own package [composer/ca-bundle](https://github.com/composer/ca-bundle)⁹, that solves this problem too. And since Composer has bigger user base, it gained traction and is now defacto a standard. Therefore Kdyby\CurlCaBundle is now deprecated. But since a lot of applications still depend on it, I am maintaining the cron job and keeping the package alive until everyone adopts [composer/ca-bundle](https://github.com/composer/ca-bundle).

3.2.12 Autowired

Kdyby\Autowired was an experiment with DI in Nette presenters. Nette presenters are similar to Controllers in Model–View–Controller

8. [guzzlehttp/guzzle](https://github.com/guzzle/guzzle) <https://github.com/guzzle/guzzle>

9. [composer/ca-bundle](https://github.com/composer/ca-bundle) <https://github.com/composer/ca-bundle>

(MVC) pattern, but have slightly different responsibilities. As such, they accept services through DI and then process the application request. Since presenters in Nette are usually part of big inheritance tree, their parents might have several dependencies that the children would have to pass through their constructors to the parents introducing what we were calling constructor hell.

Kdyby\Autowired helps with the problem by allowing to declare the dependencies in presenter properties with a special annotation. These dependencies would not have to be passed through constructor, because they are initialized lazily when they are accessed.

This works reliably, but breaks encapsulation of the presenter classes. The right solution to this problem is to have more lightweight UI\Presenter, with fewer responsibilities.

The second issue Kdyby\Autowired solves is fetching User Interface (UI) component factories that create the instance of the UI component and passing them into presenter component factories that configure the UI component for the specific use-case.

Autowiring of the factories doesn't break encapsulation of the presenter class, but it breaks the DI principle, by not exposing the direct dependencies of the presenter class, but instead it makes the presenter depend on DIC which is considered an anti-pattern.

3.2.13 FormsReplicator

Forms component of Nette has a lot of capabilities, but it does not support repeating an input field, or group of them, dynamically. This means the application is very strict about what fields it accepts when a form is submitted in browser. That is generally good for security, but sometimes it is required to accept dynamic amount of fields when the form is expected to be modified on the client and the application does not know ahead how many fields will be sent by the user. Kdyby\FormsReplicator solves this by creating the form using the data the user sent.

This library worked perfectly with Nette 2.1 but the Forms component was refactored heavily since then and the assumptions this package worked on were broken. This means there are a lot of bugs when it is used with newer Nette.

3.2.14 Translation

Translating the application UI and the content itself are two very different problems. Kdyby\Translation solves the first problem. It integrates Symfony Framework Translation Component (Symfony Translation) into Nette.

In the past, the Symfony Bundle for integrating Symfony Translation the did contain few important parts, that Kdyby\Translation has to duplicate, but not anymore. The 3.0 version provides all required functionality and the duplication can now be removed from Symfony Translation.

3.2.15 Validator

Kdyby\Validator is very thin integration of Symfony Framework Validator Component (Symfony Validator) into Nette.

3.2.16 RabbitMq

Kdyby\RabbitMq integrates php-amqp/lib/php-amqp/lib into Nette and php-amqp/lib/php-amqp/lib is a client for RabbitMq server. Kdyby\RabbitMq provides producer classes, that are used to generate messages for the RabbitMq server and consumer classes, that listen for incoming messages from the RabbitMq server and processes them with user-defined logic. It depends on Kdyby\Console, since the consumers are started as standalone processes through CLI. The php-amqp/lib/php-amqp/lib had supported old PHP versions for very long and now it is hard to change the internal design of it and Kdyby\RabbitMq hides that using abstraction.

The php-amqp/lib/php-amqp/lib changed maintainers and it has to be reviewed for new functionality, that might benefit the users of Kdyby\RabbitMq. There are also important features like [SSL connection support](#)¹⁰ that were never implemented.

10. SSL connection support <https://github.com/Kdyby/RabbitMq/issues/5>

3.2.17 Money

PHP does not have a native Decimal type, only float. And floats cannot be used to represent money values. Libraries exist that solve this exact problem. One level of abstraction above that are money libraries, that use the decimal implementations to represent the money values paired with currency.

Kdyby\Money is meant to be simpler and have more clean architecture than other competitive implementations. It operated with cent values in integers to avoid float rounding problems.

It is now deprecated, because it was really hard to maintain and there were bugs with rounding even with the integers implementation.

3.2.18 DoctrineMoney

Kdyby\DoctrineMoney integrates Kdyby\Money into Kdyby\Doctrine. Since Kdyby\Money was deprecated, this package is now deprecated too.

3.2.19 Aop

Kdyby\Aop is a custom implementation of AOP for Nette DIC component. It searches the aspect definitions, finds the services they are supposed to advice and creates proxy classes from them. In the proxy classes, it overrides the methods and applies the aspect advices.

This package was created for Nette 2.1 and Nette DIC component internals were refactored heavily since. This means the package has to be review for possible problems and updated to current libraries.

3.2.20 Clock

Logic in application often depends on current time, but if you want to unit test such logic, it is a problem. Kdyby\Clock solves this problem by implementing a DateTimeProvider service for Nette. The service with time sensitive logic should require the DateTimeProvider interface and every time it requires to know the current date and time it calls a `getTime()` method on the provider. With the dependency clearly declared in constructor of the service, it can be easily mocked and unite tested.

The package works well, but it would be useful to have such functionality framework agnostic. Now the package depends on Nette.

3.2.21 Redis

Kdyby\Redis integrates the [Pec1](#)¹¹ extension [Redis](#)¹² into Nette. There are three main services it provides - cache, cache journal, sessions and locking.

Redis is fast memory key-value storage and as such is a great candidate for cache storage. Nette contains `IStorage` interface, that Kdyby\Redis implements and it can be used as a drop-in replacement for the default file caching.

Journal in Nette is used for keeping track of tags assigned to cache values. This metadata can be then used to invalidate only specific keys in cache. The default journal in Nette was historically implemented using custom binary file and now is implemented as an SQLite database. The binary file implementation was limited by the format and could only store fixed amount of key and value pairs which is perfectly fine up to a certain point of application growth. Now the current solution with SQLite database does not have this problem, but still it requires a local file system to work. Kdyby\Redis implementation uses native Redis data structures to implement the journaling and can be used as a replacement, if the other implementations are not ideal for the application.

Session handler takes care of user sessions. In order to implement login functionality, web application has to assign session cookies to visitors and the session cookie value is used as key for session storage allowing the website to store information per user. The default implementation in PHP uses file system and is prone to failure with high traffic, because its of garbage collection mechanism. The default Redis PHP extension has its own session handler, but this handler does not handle locking of the sessions, which means a high chance for a race condition where older application request might rewrite data from newer application request. This is acceptable, if the application doesn't store any additional data in the session, apart from the user id. But

11. Pec1 <http://php.net/manual/en/install.pec1.intro.php>

12. Redis <https://pec1.php.net/package/redis>

typical Nette application stores at least flash messages in sessions and its undesirable for them to get lost.

Redis server does not have any form of native locking. It only recommends the locking algorithm [Redlock](#)¹³ and Kdyby\Redis implements it. It is used for locking cache keys when the IStorage from Nette is required to block and it is also used for the custom session handler.

Kdyby\Redis contains thin abstraction over the Redis PHP extension that is probably not necessary for the caching, journal and session services.

3.2.22 Monolog

Kdyby\Monolog is an integration of [monolog/monolog](#)¹⁴ library that implements PSR-3 into Nette. Nette has a debugging component Tracy, that renders exceptions in comprehensive way. Kdyby\Monolog handles integration with Tracy as well, making every exception passed to Monolog render with every relevant detail.

The integration works well even with current Nette, but has a lot of dead code for providing backwards compatibility, that can be dropped when the version requirement for Nette is increased. Also the configuration structure is suboptimal. The configuration of Symfony bundle for Monolog is better and Nette would benefit from having the same structure.

3.2.23 ElasticSearch

There are two main client packages for Elasticsearch - [ruflin/elastica](#)¹⁵ and [elasticsearch/elasticsearch](#)¹⁶. Kdyby\ElasticSearch integrates ruflin/elastica into Nette. The 1.* versions of elasticsearch/elasticsearch were tightly coupled with 3rd party DIC which lead to choosing the ruflin/elastica package. The main benefit of this package is the Tracy panel that prints executed queries which allows for easier debugging.

13. Redlock <https://redis.io/topics/distlock>

14. monolog/monolog <https://github.com/Seldaek/monolog>

15. ruflin/elastica <https://github.com/ruflin/Elastica>

16. elasticsearch/elasticsearch <https://github.com/elastic/elasticsearch-php>

There are several new major versions of both libraries. The Tracy panel and the bridging has to be review for bugs and possible improvements.

3.2.24 DoctrineSearch

Kdyby\DoctrineSearch integrates [doctrine/search](#)¹⁷ into Nette and extends it for a specific use-case we required at Rohlik.

3.2.25 Geocoder

Kdyby\Geocoder extends [willdurand/geocoder](#)¹⁸ with custom geocoder provider for [Seznam maps](#)¹⁹, logging capabilities and mechanism for sorting the results.

The geocoder declares a common interface for communication with maps data API providers. There are two main tasks - geocoding and reverse geocoding. Geocoding is a search in maps data for given string that returns list of addresses with GPS coordinates that correspond to the search query. Reverse geocoding takes a GPS coordinate and returns the address that corresponds to it. A geocoder provider is implementation of the common geocoding interface for the specific maps data provider.

The package now solves 4 different problems and is tightly coupled with old versions of Nette, but is well covered with unit tests.

3.2.26 CsobPaymentGateway

The czech ČSOB bank has created its own payment gateway for paying with debit and credit cards that can be used with any web or mobile application. Kdyby\CsobPaymentGateway is a client that wraps the API communication in OO abstraction. The main concerns are security and simplicity of use, since this library is part of the most critical application infrastructure - the payment process.

At the same month this library was published a competitive package [slevomat/csob-gateway](#)²⁰ was released. It is impossible to support

17. doctrine/search <https://github.com/doctrine/search>

18. willdurand/geocoder <https://github.com/geocoder-php/Geocoder>

19. Seznam maps <https://api.mapy.cz/>

20. slevomat/csob-gateway <https://github.com/slevomat/csob-gateway>

package this critical without having it in production and being able to monitor it. Since I have left the company we needed this integration for, I did not have a chance to deploy the package to any new project that I would maintain and the new versions and features of the ČSOB payment gateway are not implemented. This makes the library dangerous to use which lead to a [search for new maintainer of Kdyby\CsobPaymentGateway](#)²¹.

3.2.27 CsobPaygateNette

Kdyby\CsobPaygateNette integrates Kdyby\CsobPaymentGateway into Nette and provides a UI component for easier implementation of payment handling logic.

3.2.28 Wkhtmltopdf

[Wkhtmltopdf](#)²² is, among others, a PDF generator. It has CLI interface that accepts a HTML file, set of options and generates a PDF. Kdyby\Wkhtmltopdf is a OO abstraction of the CLI for Nette. It contains a lot of custom solutions for the subproblems, that are not easy to maintain. There are also several new versions of the tool.

3.2.29 FakeSession

Nette is first of all MVC framework which means most applications written in it depend on PHP sessions heavily. Having an abstraction that disables the mechanism but does not break application in the context of a single request is useful for writing integration tests that would otherwise interfere with each other and making sure REST API does not leak any cookies even with programmer mistake would cause it do so. Kdyby\FakeSession replaces the default session mechanism in Nette with custom one, that operates only in the memory of the current PHP process.

It is working well even with current versions of Nette but it has only few integration tests and no unit tests.

21. search for new maintainer of Kdyby\CsobPaymentGateway <https://github.com/Kdyby/CsobPaymentGateway/issues/25>

22. Wkhtmltopdf <https://wkhtmltopdf.org/>

3.2.30 RequestStack

Nette is designed to operate in the context of a single HTTP request that maps on a single application request. Therefore it has `Http\Request` object as a shared service, that any service registered into DIC can request and depend on. If one would require to handle two separate application requests, references of `Http\Request` object from the first request would already be permanently bound to all the services in DIC. `Kdyby\RequestStack` replaces the service implementation of a value object for a container of the `Http\Request` that delegates the method calls to it which means all the services have references for the `RequestStack` service that never changes and more request can be handled by simply switching the `Http\Request` reference inside the container. A perfect solution would be to redesign Nette internals to handle this problem correctly, but that would be a big architecture challenge to not break the compatibility with almost every application.

`Kdyby\RequestStack` has very few unit tests but is very simple. There have been changes in the way the related services are configured in Nette and the integration requires a small refactoring to account for this.

3.2.31 StrictObjects

PHP itself is very tolerant about object properties. It is possible to assign value to a property, that is not defined on the class of the object and it works correctly. This dynamic behavior might be useful, but has no place in serious applications since it introduces a big potential for errors. The more strict the application is about types, the safer it usually is. `Kdyby\StrictObjects` introduces a trait which is glorified mechanism for copy and paste of code between objects that fixes this behavior. PHP allows classes to implement magic methods that are called when undefined property or method is accessed. The `Kdyby\StrictObjects` trait defines them and throws exception any time you try to access undefined method or property of the class.

The package is stable, does not depend on Nette and the only problem it has at this moment is no support for PHP 7.1.

3.2.32 Facebook

With applications that need to interact with Facebook there are two main use-cases - Facebook OAuth login and advertisement data manipulation. Both require secure communication with the Facebook Graph API. Kdyby\Facebook supports the OAuth 2 login mechanism of Facebook and provides a client for the Facebook Graph API communication.

Facebook keeps releasing new version of their API that keeps breaking backwards compatibility which results in a lot of maintenance work of Kdyby\Facebook. But the package did not receive enough attention and the last supported version of Facebook Graph API is v2.3 which is supported only until 2017-08-07 and as of writing this the current [version of Facebook Graph API](https://developers.facebook.com/docs/apps/changelog)²³ is v2.9.

3.2.33 Google

Kdyby\Google is an integration of [google/apiclient](https://github.com/google/google-api-php-client)²⁴ which aims to solve primarily OAuth 2 login functionality for Nette. It was written with old PHP 5.3, users report it cannot be even easily installed.

3.2.34 Github

Kdyby\Github is a custom implementation of a Github API client and its integration into Nette. It enables secure authentication when its UI component is used.

3.2.35 BootstrapFormRenderer

The CSS framework [Twitter Bootstrap](http://getbootstrap.com/)²⁵ has a specific markup of HTML forms that is very different from what Nette Forms component generates by default. This package implements a custom Forms renderer that enables simpler markup and often completely automatic rendering of most forms a typical application might have. The package was relevant for Twitter Bootstrap 2.2 and Nette 2.1. New versions of

23. version of Facebook Graph API <https://developers.facebook.com/docs/apps/changelog>

24. google/apiclient <https://github.com/google/google-api-php-client>

25. Twitter Bootstrap <http://getbootstrap.com/>

Nette have more features enabling them to be configured to render the required markup by default instead of having to replace the renderer. The package is currently abandoned and unmaintained.

3.2.36 PayPalExpress

PayPal is popular payment platform and Kdyby\PayPalExpress provided a custom API client and its integration into Nette. Now is deprecated because I had no use for it in any production application and no new maintainer was found which means its dangerous to invite other people to use it.

3.2.37 PresentersLocator

There is a `PresenterFactory` in Nette that is responsible for mapping the application request details to appropriate presenter class and then creating instance of it. In current versions of Nette the `PresenterFactory` delegates the creation of the instance to the DIC but it was not doing it in older versions of Nette. Kdyby\PresentersLocator is automatically discovering presenter classes in the application and registering them as services in compile-time of the DIC. Since Nette currently solves this problem without the need for this extension, it was deprecated.

4 Designing roadmap of refactoring

This chapter contains overview of the plan for refactoring. 4.1 explains what packages will be deprecated and why. 4.2 evaluates common requirements for all the packages, that will undergo refactoring. Finally 4.3 describes specific plan for the individual packages, if there are any.

4.1 Deprecations

No new maintainer for Kdyby\CsobPaymentGateway and Kdyby\CsobPaygateNette was found and therefore it would be dangerous to encourage usage of these packages in such critical piece of infrastructure without having a maintainer that uses them in production. They are both going to be deprecated. They are well tested and work correctly with the versions of ČSOB payment gateway and Nette they are written for. The users of these packages will have to migrate to [slevomat/csob-gateway](https://github.com/slevomat/csob-gateway)¹.

4.2 Common requirements

Every package is specific, but there is set of standards that have to be enforced for all good OSS PHP projects and it is constantly evolving. Having only tests and documentation is no longer the only best practice. The following sections cover the requirements.

4.2.1 Static analysis with PHPStan

[PHPStan](https://github.com/phpstan/phpstan)² claims to be a static analysis tool that can discover bugs in source code. It does not find all bugs, but it has a lot of helpful checks. Running such tool on Kdyby packages in CI is going to increase confidence in the correctness of the packages and help prevent introducing new bugs.

But there is a problem with PHPStan - it is not really a static analysis tool. It does analyze the code, but not statically. It uses two tools for its

1. [slevomat/csob-gateway](https://github.com/slevomat/csob-gateway) <https://github.com/slevomat/csob-gateway>

2. [PHPStan](https://github.com/phpstan/phpstan) <https://github.com/phpstan/phpstan>

analysis - [PHP Parser](#)³ and [native PHP reflection](#)⁴. First it parses the source code using the PHP Parser and when it finds a class [PHPStan](#) [loads the file into memory and executes it](#)⁵ which makes it available for the PHP reflection. This would not be a problem on itself with source code that has no side effects but PHPStan has 3rd party dependencies that might clash with dependencies of the project it is analyzing. One of them is Symfony Console. If PHPStan was to analyze the source code of Symfony Console it would not be analyzing the source code of the library it would be executed on, but the source code of its own dependency, because it uses the native PHP reflection.

This problem has several solutions. PHPStan can be rewritten to not use the native PHP reflection, but emulated one that works on top of the PHP Parser. The author does not like this solution because he is worried about speed of the tool. PHPStan can be rewritten to drop all dependencies and re-implement the functionality the libraries provide. This solution is not good because it would create additional and unnecessary overhead for the maintainers. Or we can implement a compiler that preprocesses the source code of PHPStan and its dependencies and fixes the problem.

Citing the PHP documentation, PHAR provides a way to put entire PHP applications into a single file called a PHP Archive for easy distribution and installation [26]. PHPStan has also opened an issue [PHAR file for each release](#)⁶ where community is requesting releases to be made also in PHAR.

I am going to implement a compiler that fixes the problem with type collisions and creates a PHAR distribution of the tool. I am also going to offer the author to take over the project afterwards so he can make it official part of the PHPStan ecosystem.

3. PHP Parser <https://github.com/nikic/PHP-Parser>

4. native PHP reflection <http://php.net/manual/en/book.reflection.php>

5. PHPStan loads the file into memory and executes it <https://github.com/phpstan/phpstan/issues/137>

6. PHAR file for each release <https://github.com/phpstan/phpstan/issues/110>

4.2.2 Coding Standard

4.3 Specific requirements for each package

4.3.1 Doctrine

Lorem ipsum.

4.3.2 Console

Lorem ipsum.

4.3.3 Events

Lorem ipsum.

4.3.4 Annotations

Lorem ipsum.

4.3.5 DoctrineCache

Lorem ipsum.

4.3.6 DoctrineMagicAccessors

Lorem ipsum.

4.3.7 DoctrineCollectionsReadonly

Lorem ipsum.

4.3.8 DoctrineCollectionsLazy

Lorem ipsum.

4.3.9 DoctrineDbalBatchImport

Lorem ipsum.

4.3.10 DoctrineForms

Lorem ipsum.

4.3.11 Autowired

Lorem ipsum.

4.3.12 FormsReplicator

Lorem ipsum.

4.3.13 Translation

Lorem ipsum.

4.3.14 Validator

Lorem ipsum.

4.3.15 RabbitMq

Lorem ipsum.

4.3.16 Money

Lorem ipsum.

4.3.17 DoctrineMoney

Lorem ipsum.

4.3.18 Aop

Lorem ipsum.

4.3.19 Clock

Lorem ipsum.

4.3.20 Redis

Lorem ipsum.

4.3.21 ParseUseStatements

Lorem ipsum.

4.3.22 RedisActiveLock

Lorem ipsum.

4.3.23 TesterParallelStress

Lorem ipsum.

4.3.24 Monolog

Lorem ipsum.

4.3.25 ElasticSearch

Lorem ipsum.

4.3.26 DoctrineSearch

Lorem ipsum.

4.3.27 Geocoder

Lorem ipsum.

4.3.28 CsobPaygateNette

Lorem ipsum.

4.3.29 CsobPaymentGateway

Lorem ipsum.

4.3.30 Wkhtmltopdf

Lorem ipsum.

4.3.31 FakeSession

Lorem ipsum.

4.3.32 RequestStack

Lorem ipsum.

4.3.33 StrictObjects

Lorem ipsum.

4.3.34 Facebook

Lorem ipsum.

4.3.35 Google

Lorem ipsum.

4.3.36 Github

Lorem ipsum.

4.3.37 NettePhpServer

Lorem ipsum.

4.3.38 TesterExtras

Lorem ipsum.

4.3.39 HtmlValidatorPanel

Lorem ipsum.

5 The refactoring process of Kdyby

This chapter documents what I have accomplished with each package in detail.

5.1 Doctrine

Lorem ipsum.

5.2 Console

Lorem ipsum.

5.3 Events

Lorem ipsum.

5.4 Annotations

Lorem ipsum.

5.5 DoctrineCache

Lorem ipsum.

5.6 DoctrineMagicAccessors

Lorem ipsum.

5.7 DoctrineCollectionsReadonly

Lorem ipsum.

5.8 DoctrineCollectionsLazy

Lorem ipsum.

5.9 DoctrineDbalBatchImport

Lorem ipsum.

5.10 DoctrineForms

Lorem ipsum.

5.11 Autowired

Lorem ipsum.

5.12 FormsReplicator

Lorem ipsum.

5.13 Translation

Lorem ipsum.

5.14 Validator

Lorem ipsum.

5.15 RabbitMq

Lorem ipsum.

5.16 Money

Lorem ipsum.

5.17 DoctrineMoney

Lorem ipsum.

5.18 Aop

Lorem ipsum.

5.19 Clock

Lorem ipsum.

5.20 Redis

Lorem ipsum.

5.21 ParseUseStatements

Lorem ipsum.

5.22 RedisActiveLock

Lorem ipsum.

5.23 TesterParallelStress

Lorem ipsum.

5.24 Monolog

Lorem ipsum.

5.25 ElasticSearch

Lorem ipsum.

5.26 DoctrineSearch

Lorem ipsum.

5.27 Geocoder

Lorem ipsum.

5.28 CsobPaygateNette

Lorem ipsum.

5.29 CsobPaymentGateway

Lorem ipsum.

5.30 Wkhtmltopdf

Lorem ipsum.

5.31 FakeSession

Lorem ipsum.

5.32 RequestStack

Lorem ipsum.

5.33 StrictObjects

Lorem ipsum.

5.34 Facebook

Lorem ipsum.

5.35 Google

Lorem ipsum.

5.36 Github

Lorem ipsum.

5.37 NettePhpServer

Lorem ipsum.

5.38 TesterExtras

Lorem ipsum.

5.39 HtmlValidatorPanel

Lorem ipsum.

6 Conclusion and future work

We made it!

Bibliography

- [1] Wikipedia. *Open-source software* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Open-source_software&oldid=776201239.
- [2] Wikipedia. *PHP* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=PHP&oldid=775984544>.
- [3] *Packages from kdyby - Packagist*. 2017. URL: <https://packagist.org/packages/kdyby/>.
- [4] Wikipedia. *Content management system* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Content_management_system&oldid=775559667.
- [5] Wikipedia. *Free software license* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Free_software_license&oldid=776336813.
- [6] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: <https://martinfowler.com/articles/injection.html>.
- [7] Wikipedia. *Aspect-oriented programming* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Aspect-oriented_programming&oldid=751829822.
- [8] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014. ISBN: 1484200772.
- [9] Wikipedia. *Continuous integration* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 28-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Continuous_integration&oldid=777232210.
- [10] Wikipedia. *Nette Framework* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Nette_Framework&oldid=766339218.
- [11] Wikipedia. *Package manager* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Package_manager&oldid=775128084.

-
- [12] Composer Community. *Introduction - Composer*. 2017. URL: <https://getcomposer.org/doc/00-intro.md>.
 - [13] Wikipedia. *JSON* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=JSON&oldid=775860167>.
 - [14] David Grudl. *Dibi is Database Abstraction Library for PHP 5 - Homepage*. 2017. URL: <https://dibiphp.com/>.
 - [15] Wikipedia. *Object-relational mapping* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Object-relational_mapping&oldid=769454010.
 - [16] Wikipedia. *Object-oriented programming* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=775231020.
 - [17] Wikipedia. *Symfony* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=Symfony&oldid=769598534>.
 - [18] Wikipedia. *RabbitMQ* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=RabbitMQ&oldid=774630105>.
 - [19] Wikipedia. *Elasticsearch* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=Elasticsearch&oldid=776540688>.
 - [20] Wikipedia. *Redis* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: <https://en.wikipedia.org/w/index.php?title=Redis&oldid=774326024>.
 - [21] Ondřej Mirtes. *PHPStan - PHP Static Analysis Tool*. 2017. URL: <https://github.com/phpstan/phpstan>.
 - [22] PHP Framework Interop Group. *PHP Standards Recommendations*. 2017. URL: <http://www.php-fig.org/psr/>.
 - [23] Jordi Boggiano. *Monolog - Logging for PHP*. 2017. URL: <https://seldaek.github.io/monolog/>.
 - [24] Wikipedia. *Unit testing* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Unit_testing&oldid=773620225.
 - [25] David Grudl. *Nette Tester – enjoyable unit testing*. 2017. URL: <https://tester.nette.org/en/>.

BIBLIOGRAPHY

- [26] PHP community. *PHP Manual - Function Reference - Compression and Archive Extensions - Phar*. 2017. URL: <http://php.net/manual/en/intro.phar.php>.