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

I would like to thank the Kdyby community that kept using the packages even through I had no time to maintain them for almost year and a half. To my advisor RNDr. Jaroslav Bayer for helpful insights and inspiring conversations. To my roommates and friends for surviving the constant whining about not getting enough sleep and having drank too much coffee. To my girlfriend and family for not letting me give up.

And finally big thanks to my computer for surviving it till the end, even though for few days it looked like that my new SSD was dying again.

Abstract

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

Keywords

Kdyby, package, Packagist, Composer, Nette Framework, Doctrine ORM, ORM, Symfony Framework, integration, refactoring, clean code, coding style, coding standard, PHP

Contents

1 Introduction

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

Through my career, 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 into their applications as well.

As of writing this, the more successful libraries have hundreds of thousands of downloads. Five of Kdyby libraries have over quarter million downloads, and one is approaching half a million with a staggering amount of 497 thousands of downloads[1]. In conclusion, a sober estimate would be, that hundreds of real production applications use Kdyby libraries.

If I account only for the two biggest projects that I can confirm are using Kdyby packages, over a billion Czech crowns[2] has flown 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 want to use this thesis as a way to fix the situation.

The primary goal of this thesis is to resolve compatibility of Kdyby packages with new versions of libraries they integrate with, fix as many bugs as possible and increase the quality of their code. I will also review the state of each library and decide its future. I will either deprecate it and provide the users a suggestion for a better alternative, or fix the problems and refactor the library.

---

1. https://packagist.org/packages/kdyby/
2. 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). 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 was introduced to me, and I have decided to start working on everything openly, under a free license [2]. No new working version of Kdyby CMS was released since then because I have rewritten it from scratch exactly ten times.

In 2012, I decomposed the entire 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 [3].

2.2.2 Aspect Oriented Programming

Complex systems have to deal with a lot of other problems that are not the center of the business logic, but also important. This might be for example transaction handling or logging. Aspect–oriented Programming (AOP) is a paradigm that aims to increase modularity by helping to separate these cross-cutting concerns. It allows changing

1. https://github.com/fprochazka/kdyby-cms-old

behavior of existing code without modifying the code itself using advices that are applied using pointcut specification [4]. A good example is a `Transactional` annotation in Java world. A pointcut is defined to bind to every method with such annotation and the advice wraps the method in a transaction.

2.2.3 Observer pattern

Observer a pattern for writing modular code. It allows creating 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 [5].

Symfony provides an implementation called Symfony Framework EventDispatcher Component (Symfony EventDispatcher) that the subscriber implementations can listen on.

2.3 Technologies used

2.3.1 Git and GitHub

Git is a Version Control System that is decentralized and considered very fast [6]. 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 aid 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 [7]. But nowadays, 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[2], CircleCI[3] and GitLab CI[4]. Kdyby is using the Travis CI that is free for OSS projects.

2.3.3 Semantic Versioning

Semantic Versioning (SemVer) is a standard that defines how software should be versioned to allow safe upgrading. An application might be written with a particular release of the library and upgrading to the newest version might break it because of dropped compatibility. SemVer[5] defines major, minor and patch releases that signal what versions are compatible with each other.

2.3.4 Nette Framework

Nette Framework (Nette) is an OSS framework for creating web applications in PHP. Nette is separated into many packages [8].

The DI component nette/di[6] provides a Dependency Injection Container (DIC) that holds the services. The component also allows preparing a preconfigured DIC, which is then compiled into a PHP

2. https://travis-ci.org/
3. https://circleci.com/
4. https://about.gitlab.com/features/gitlab-ci-cd/
5. http://semver.org/
6. https://github.com/nette/di

class that contains optimized code for the service creation. This compiled DIC 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 Nette 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.

Default settings for error handling in PHP are not ideal. It outputs errors and warnings directly into the browser for the user to see which is bad for esthetical, usability and security reasons. Tracy[7] provides error and exception handlers that redirect the messages into log files. It also renders a red BlueScreen HTML file with extensive details about the error that occurred. The BlueScreen renderer is extensible and allows installing custom panels that can provide additional context for the programmer.

2.3.5 Composer

Composer is a tool for dependency management in PHP. It allows you to declare the libraries your project depends on and it will manage (install or update) them for you [9].

Packages are usually published using GitHub with metadata in a file named `composer.json` that is written in JSON format.

Composer is decentralized but has a single main metadata repository Packagist[8]. 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.

2.3.6 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 essen-

7. https://github.com/nette/tracy
8. https://packagist.org/

tially enabling them to login or register to services through the OAuth 2 provider with two clicks without revealing their password [10].

Kdyby provides packages for integrating Nette with OAuth 2 providers, such as Facebook[9], Google[10] and GitHub[11].

2.3.7 Doctrine 2 ORM

Doctrine 2 ORM is an Object–Relation Mapper (ORM), which means it allows the programmer to create PHP classes called entities that represent relational data in a database and are used to map the data from the database to the classes and back. In conclusion, it allows the programmer to write fully Object-Oriented (OO) applications [11].

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 [12].

For extensibility, Symfony has Bundles that provide similar functionality to Nette `CompilerExtension`, but they operate on a different level of abstractions. A 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 a messaging broker software (sometimes called message-oriented middleware) that implements Advanced Message Queuing Protocol (AMQP) and other protocols. Messages can be routed into many queues that consumers listen on. It also provides mirroring across a cluster of machines increasing availability in case of hardware failure [13].

---

9. https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow
10. https://developers.google.com/identity/protocols/OAuth2
11. https://developer.github.com/v3/oauth/

2.3.10 ElasticSearch

Elasticsearch is a real–time search and analytics engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with a coherent RESTful API and schema–free JSON documents. Elasticsearch is developed in Java and is released as OSS under the terms of the Apache License. It is the most popular enterprise search engine today [14].

2.3.11 Redis

Redis is an OSS in–memory data structure store that can be used as database, cache or messaging broker. It supports a variety of basic data structures, geospatial indexes, and others. It has a replication, transactions, LUA scripting, on–disk persistence and provides high availability via Redis Cluster [15].

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 [16].

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[12] [17].

12. http://www.php-fig.org/psr/

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 logging interface that can be typehinted against in other libraries to keep maximum of interoperability [18].

2.3.15 Nette\Tester

Nette\Tester is a unit testing framework for PHP [19]. Its main advantage over other unit testing libraries is that by default every test runs in a single process and the tests are executed in parallel making them run faster, in better isolation, and they cannot depend on a specific order of execution.

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 made it to production, and at least one 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 compatibility with new versions of 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 2.1 or older.

The other problems only appear when you interact with the source code which is still crucial for me as a maintainer and for the contributors. Good maintainable code attracts more programmers to use it and contribute. 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 the end of life at 2015-09-03[1] and is no longer supported by PHP developers.

PHP 5.5 had the end of life at 2016-06-21, and PHP 5.6 is currently in the phase of security fixes only and will have the 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 developer has to consider what PHP versions the libraries support, upgrade them first and then migrate the application.

1. https://secure.php.net/eol.php

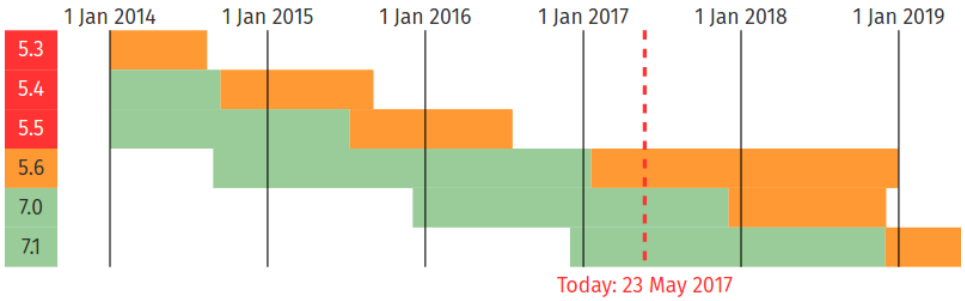

Figure 3.1: Supported PHP Versions. Green is active support, orange is security fixes only. Up-to-date version is at https://secure.php.net/supported-versions.php.

Most of the packages are compatible with HipHop Virtual Machine (HHVM), but Nette has dropped the compatibility making it impossible to maintain.

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 do not belong to it

Doctrine 2 ORM itself is separated into several packages, mainly doctrine/orm[2], doctrine/common[3], doctrine/annotations[4], doctrine/-cache[5] and 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.

2. https://github.com/doctrine/doctrine2
3. https://github.com/doctrine/common
4. https://github.com/doctrine/annotations
5. https://github.com/doctrine/cache
6. https://github.com/doctrine/collections

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 huge SQL scripts into the database.

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

New versions of Nette and Doctrine 2 ORM were released and entirely 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.

Some tasks 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* is now causing more problems than it solves.

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 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. https://github.com/Kdyby/Doctrine/issues/238

```php
/**
 * @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 *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 does not have native annotations. But Doctrine simulates them using PHPDoc. The example code 3.2 illustrates the 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 a few minor problems and inefficiencies that have to be fixed to work correctly with current Nette and Doctrine.

3.2.6 DoctrineMagicAccessors

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

In Doctrine 2 ORM there is an entity lazy–loading feature that 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 a collection of order items. Adding an order item to the cart might be done using an `addOrderItem` method or through the collection directly as shown in 3.2.

```php
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.

```php
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 accessed.

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

3.2.9 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 OO API.

There are now better and more attractive packages for performing HTTP requests in PHP like guzzlehttp/guzzle[8]. Therefore this package is now deprecated and unmaintained.

8. https://github.com/guzzle/guzzle

3.2.10 CurlCaBundle

For executing 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 a format that the Curl clients can use and then publishes them as a new version of *Kdyby\CurlCaBundle*.

Using this package in an 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[9] that solves this problem too. And since Composer has bigger user base, it gained traction and is now de facto 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.

3.2.11 Autowired

Kdyby\Autowired was an experiment with DI in Nette presenters. Nette presenters are similar to Controllers in Model–View–Controller (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 a 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 particular annotation. These dependencies would not have to be passed through a constructor because they are lazily initialized when they are accessed.

---

9. https://github.com/composer/ca-bundle

This works reliably but breaks the 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 particular use–case.

Autowiring of the factories does not 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.

3.2.12 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 a browser. That is good for security, but sometimes it is required to receive 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\Forms-Replicator* solves this by creating the form using the data the user sent.

This library worker 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.

3.2.13 Translation

Translating the application UI and the content itself are two very different problems. *Kdyby\Translation* solves the first issue. 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.

3.2.14 Validator

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

3.2.15 RabbitMq

Kdyby\RabbitMq integrates php-amqplib/php-amqplib[10] into Nette and php-amqplib/php-amqplib 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-amqplib/php-amqplib 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-amqplib/php-amqplib 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[11] that were never implemented.

3.2.16 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 hard to maintain and there were bugs with rounding even with the integers implementation.

10. https://github.com/php-amqplib/php-amqplib
11. https://github.com/Kdyby/RabbitMq/issues/5

3.2.17 DoctrineMoney

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

3.2.18 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 advise and creates proxy classes from them. In the proxy classes, it overrides the methods and applies the aspect advice.

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

3.2.19 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 issue by implementing a `DateTimeProvider` service for Nette. The service with time sensitive logic should require the `DateTimeProvider` interface and every time it needs to know the current date and time it calls a `getTime()` method on the provider. With the dependency explicitly declared in the constructor of the service, it can be easily mocked and unit tested.

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

3.2.20 Redis

Kdyby\Redis integrates the Pecl[12] extension Redis[13] 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 an excellent candidate for cache storage. Nette contains an `IStorage` interface that *Kdyby\Redis* implements and it can be used as a drop-in replacement for the default file caching.

12. https://secure.php.net/manual/en/install.pecl.intro.php
13. https://pecl.php.net/package/redis

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 the 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 utilized as a replacement if the other implementations are not ideal for the application.

Session handler takes care of user sessions. A web application has to assign session cookies to visitors to implement login functionality, and the session cookie value is used as a 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 of its garbage collection mechanism. The default Redis PHP extension has its session handler, but this handler does not handle locking of the sessions, which means a high chance for a race condition where earlier application request might rewrite data from newer application request. This is acceptable if the application does not store any additional data in the session, apart from the user id. But regular 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[14] 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 utilized 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.

14. https://redis.io/topics/distlock

3.2.21 Monolog

Kdyby\Monolog is an integration of monolog/monolog[15] library that implements PSR-3 into Nette. *Kdyby\Monolog* handles integration with Tracy as well, making every exception passed to Monolog render with every relevant detail using Tracy BlueScreen renderer.

The integration works well even with current Nette but has a lot of dead code for providing backward 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 than what *Kdyby\Monolog* has now and Nette would benefit from having the same structure.

3.2.22 ElasticSearch

There are two main client packages for Elasticsearch - ruflin/elastica[16] and elasticsearch/elasticsearch[17]. *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 allow for easier debugging.

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

3.2.23 DoctrineSearch

Kdyby\DoctrineSearch integrates doctrine/search[18] into Nette and extends it for a specific use–case we required at Rohlik.

15. https://github.com/Seldaek/monolog
16. https://github.com/ruflin/Elastica
17. https://github.com/elastic/elasticsearch-php
18. https://github.com/doctrine/search

3.2.24 Geocoder

Kdyby\Geocoder extends willdurand/geocoder[19] with custom geocoder provider for Seznam maps[20], 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.

3.2.25 CsobPaymentGateway

The Czech ČSOB bank has created its 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[21] was released. It is impossible to support 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 a new maintainer of *Kdyby\CsobPaymentGateway*[22].

19. https://github.com/geocoder-php/Geocoder
20. https://api.mapy.cz/
21. https://github.com/slevomat/csob-gateway
22. https://github.com/Kdyby/CsobPaymentGateway/issues/25

3.2.26 CsobPaygateNette

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

3.2.27 Wkhtmltopdf

Wkhtmltopdf[23] is, among others, a PDF generator. It has CLI interface that accepts an HTML file, set of options and generates a PDF. *Kdyby\Wkhtmltopdf* is an 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.

3.2.28 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 the 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 a 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.

3.2.29 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 it. If one would require handling 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

23. https://wkhtmltopdf.org/

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 just 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 not to 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.

3.2.30 StrictObjects

PHP itself is very tolerant about object properties. It is possible to assign a 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 significant potential for errors. The more strict the application is about types, the safer it usually is. *Kdyby\StrictObjects* provides a trait which is a 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 an 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.

3.2.31 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 versions of its API that keeps breaking backward 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[24] is `v2.9`.

3.2.32 Google

Kdyby\Google is an integration of google/apiclient[25] which aims to solve OAuth 2 login functionality primarily for Nette. It was written with old PHP 5.3. Users report it cannot be even easily installed.

3.2.33 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.

3.2.34 BootstrapFormRenderer

The CSS framework Twitter Bootstrap[26] 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 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.

3.2.35 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 it is dangerous to invite other people to use it.

---

24. https://developers.facebook.com/docs/apps/changelog
25. https://github.com/google/google-api-php-client
26. http://getbootstrap.com/

4 Roadmap of refactoring

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

The roadmap extends the scope of this thesis and documents even some changes that are planed in the future.

4.1 Deprecations

Deprecating a package means it will be visibly marked on GitHub as not maintained anymore and on Packagist there is a particular feature for abandoning packages. When somebody tries to install abandoned the package, Composer will show a warning that the package is not maintained and they should migrate away from it. Deprecation is reversible, and if somebody who is using the package wants to start taking care of it, I will allow it and assign them the maintainer permissions.

New maintainer for *Kdyby\CsobPaymentGateway* and *Kdyby\CsobPaygateNette* was not found and therefore it would be dangerous to encourage the use 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[1].

Kdyby\DoctrineSearch will be deprecated, and I will make no attempts at fixing it. It is based on a prototype package, and no stable versions were released. If anybody was using it, they were doing so knowingly risking this outcome.

There will be new releases for Nette 2.3 and 2.4 of *Kdyby\Facebook*, *Kdyby\Google* and *Kdyby\Github*. But after that, the packages will be deprecated. The OAuth 2 is a standard, and all the providers adhere

1. https://github.com/slevomat/csob-gateway

to it with only slight variations. A generic library like league/oauth2-client[2] that implements the standard rather than separate integrations with each provider is more sustainable.

All Kdyby packages will drop support for HHVM. It has too low adoption to outweigh the extra maintenance work it requires and supporting HHVM on packages that depend on Nette is impossible.

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 current trends.

4.2.1 Static analysis with PHPStan

PHPStan[3] 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 necessary 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 a static analysis tool. It does analyze the code, but not statically. It uses two tools for its analysis - PHP Parser[4] and native PHP reflection[5]. 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[6] 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 were 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 dependency, because it uses the native PHP reflection.

2. https://github.com/thephpleague/oauth2-client
3. https://github.com/phpstan/phpstan
4. https://github.com/nikic/PHP-Parser
5. https://secure.php.net/manual/en/book.reflection.php
6. https://github.com/phpstan/phpstan/issues/137

This problem has several solutions. PHPStan can be rewritten not to 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 the 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 [20]. PHPStan has also opened an issue PHAR file for each release[7] where community is requesting releases to be also made 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 afterward so he can make it an official part of the PHPStan ecosystem.

4.2.2 Coding Standard with PHP_CodeSniffer

Kdyby has a coding standard from the beginning that is based on Nette coding standard, but no tool is automatically enforcing it. I have refused to use PHP_CodeSniffer[8] in the past because it does not have good architecture and did not support the rules Kdyby Coding Standard required and somebody would have to implement them first. Now there is slevomat/coding-standard[9] project that covers most of the needs Kdyby has, and it is reasonable to revisit PHP_CodeSniffer now.

Kdyby will use consistence/coding-standard[10] as a base definition. Consistence Coding Standard includes the slevomat/coding-standard rules. Kdyby Coding Standard will inherit it and modify the rules settings to account for the differences in the standard.

---

7. https://github.com/phpstan/phpstan/issues/110
8. https://github.com/squizlabs/PHP_CodeSniffer
9. https://github.com/slevomat/coding-standard
10. https://github.com/consistence/coding-standard

4.2.3 Nette 2.3 and 2.4 compatible versions

Each supported package that depends on Nette must be fixed for Nette 2.3 if that is not an unreasonable amount of extra work considering the 2.3 version is a legacy now. Otherwise, the 2.3 will be skipped. Then the minimum required version will be increased to Nette 2.4 and another fixed version will be released that will preferably drop all code that handled backward compatibility with old Nette. This will allow for less source code and will serve as a better base for future releases.

Releasing the versions that solve compatibility with current Nette versions is a more important than applying the new Coding Standard and will be prioritized.

4.2.4 PHP 5.6 and newer only

After the bugfix versions are released, all packages will drop compatibility with PHP 5.5 or older in master branch. No new feature releases will support old PHP unless there is a critical bug that will require a patch release for an older version of the package. Compatibility with PHP 7.0 and 7.1 will be fixed and enforced by CI.

All the packages have gathered some bug reports in their issue trackers. What can be fixed for Nette 2.3 or 2.4 will be fixed before that release. Everything else that requires architecture changes to fix the problems or implement new features will not be implemented before the minimum requirement of PHP 5.6 is enforced. It will also not be implemented before the package is fixed based on the PHPStan analysis and new Coding Standard is applied and enforced.

4.2.5 PHP 7.1 and newer

Kdyby packages will skip minimum requirement of PHP 7.0. After the support for PHP 5.6 is dropped the support for PHP 7.0 will be dropped with it. PHP 7.0 introduces return value typehinting and scalar typehinting, allowing to declare if argument should be string or integer which was not possible until PHP 7.0. But it is missing nullable

types[11] and void return type[12] which are both important and the new type system is incomplete without them.

PHP 7.1 releases will not be part of refactoring covered by this thesis, but they are an essential element of the roadmap and should be mentioned.

4.2.6 Framework agnostic libraries

Most of the packages are an integration of some tool into Nette, but many of them extend the functionality of the original package making them a candidate for a split into two or more packages. An excellent example of this is *Kdyby\Doctrine* that accumulated many extra features. If such package were installed into a Symfony application, it would drag along all its dependencies and tight coupling for Nette. The additional dependencies can be ignored, and it would most likely work, but that is not the best way to develop applications.

The solution to this problem is to extract the functionality from the packages that violate the Unix philosophy to do one thing only and to do it right. The extracted packages that are framework agnostic can be used with Symfony or with other PHP frameworks easily, and the Nette community will benefit from bigger potential user base which inherently makes any OSS better.

4.3 Specific requirements for each package

This section will only broadly cover the interesting and significant architectural changes that will be made and not go in depth on all the issues that have to be fixed.

4.3.1 Console

The introduction of `LinkGenerator` service in Nette resolved the problem with generating URLs. *Kdyby\Console* can, therefore, drop the entire abstraction that was fixing the issue.

---

11. https://secure.php.net/manual/en/migration71.new-features.php#migration71.new-features.nullable-types
12. https://secure.php.net/manual/en/migration71.new-features.php#migration71.new-features.void-functions

4.3.2 Events

After fixing the compatibility with Nette 2.3 and 2.4, there will be a significant change of philosophy in this package. The EventManager that *Kdyby\Doctrine* requires will be simplified and moved to a bridge package between Nette and Doctrine 2 ORM. Its only responsibility will be making sure the event subscriber classes for Doctrine 2 ORM are lazy–initialized and fetched from DIC only when they are required. The rest of the package will be deprecated, and users will be encouraged to use Symfony EventDispatcher in new projects.

4.3.3 Doctrine

Doctrine 2 ORM dropped support for older PHP in `master` branches and will have the newest releases only for PHP 7.1 and newer. After the Nette 2.3 and 2.4 versions with bugfixes are released, the master will switch the minimum dependency on PHP to 7.1 and the refactored version will be written directly in PHP 7.1 and newer.

Kdyby\Doctrine provides a Tracy panel that violates Single responsibility principle (SRP) because it solves rendering SQL queries, rendering second level cache statistics and rendering of Tracy BlueScreen panels for exceptions. The SQL Panel for Tracy will be separated to standalone package *Kdyby\TracyDoctrineDbalPanel* so it can be used with only Tracy and Doctrine with no other dependencies required. The remaining functionality will be extracted into two different classes where one renders cache statistics in Tracy panel and the other render panels with additional context for Tracy BlueScreens.

Manager class from Doctrine 2 ORM has been extended to provide extra configuration options and custom diagnostics features. But Doctrine 2 ORM internally assumes the `EntityManager` is final and should not be extended, only decorated. The inherited class can be removed because the problems it solved can be resolved without inheritance.

The structure of configuration will be changed to be roughly the same as Symfony Doctrine Bundle has. Together with that, the `CompilerExtension` will be refactored to only register services that are necessary for Doctrine 2 ORM to run correctly.

The `EntityManager` holds state and has a `close` feature that when exception rises while it is synchronizing state with the database, it

gets locked and will not perform any operations. That is perfectly reasonable, but the application must be able to recover from such state. `ManagerRegistry` solves that partially, and *Kdyby\Doctrine* already implements it. But a better solution is to utilize ocramius/proxy-manager[13] that creates a proxy for the manager and allows to reset the service completely, even when other services have already reference for the services.

A lifecycle event subscriber for a particular entity type is called entity listener. The entity listeners often have dependencies that should be injected using DI. That will be achieved with a custom implementation of entity listener resolver `ContainerAwareEntityListenerResolver` that will fetch the configured listeners from DIC.

A `ConnectionFactory` will be implemented to handle registration of custom data types for the entity fields. This is now solved directly in inherited `Doctrine\DBAL\Connection` but that violates SRP.

Custom `ContainerAwareEventManager` for lazy initialization of subscriber classes will be a part of *Kdyby\Doctrine* directly, and dependency on *Kdyby\Events* will be removed.

To be able to configure parameters for Doctrine 2 ORM filters a `ManagerConfigurator` will be implemented that will hold the parameters and pass them into the filters.

The `NonLockingUniqueInserter` for atomic inserts can be extracted into a standalone package with dependency only on Doctrine 2 ORM.

Single package will be extracted with `QueryObject` and `ResultSet` for writing self–contained Doctrine Query Language (DQL) queries.

Repositories reimplemented using DQL and extended `QueryBuilder` with auto–join feature will be extracted into a standalone package.

All the extracted packages except *Kdyby\TracyDoctrineDbalPanel* should be completely optional and framework agnostic. This will allow using them with other frameworks, not only Nette.

4.3.4 Translation

The `~3.0` releases of Symfony Translation solve problems that *Kdyby\Translation* tried to fix, and a lot of code can be removed completely.

13. https://github.com/Ocramius/ProxyManager

4.3.5 Clock

Kdyby\Clock will be separated into two packages where one would only cover implementing the `DateTimeProvider` itself, and the other package will integrate it with Nette.

4.3.6 Geocoder

Four new packages will be extracted from *Kdyby\Geocoder*, and this package will be deprecated. There will be *Kdyby\GeocoderLogging*, *Kdyby\GeocoderSeznamMaps*, *Kdyby\GeocoderGoogleMapsProxied* and *Kdyby\GeocoderBestMatch*.

4.3.7 Wkhtmltopdf

This package implements a custom process handling that will be replaced with symfony/process[14]. It also contains both the implementation of the CLI OO abstraction and the Nette integration itself. These responsibilities will be separated into individual packages.

14. https://github.com/symfony/process

5 The refactoring process of Kdyby

This chapter documents the most interesting changes that were made to Kdyby and the refactoring process itself.

5.1 Implementing PHPStan compiler

The PHPStan PHAR is an essential part of the refactoring and new CI setup. Therefore I consider the compiler I had to write an important aspect of this thesis. Few challenges have to be solved before its implementation.

This section first documents the source code preprocessing. Supporting the official PHPStan extensions is covered next. And the last step is to implement a PHAR archiver that will output a single executable PHAR file.

5.1.1 Source code preprocessor

The solution we have agreed upon with the author of PHPStan is to prefix the namespaces of the PHPStan dependencies with `PHPStanVendor\\` and fix all references to them in the whole project including other dependencies.

For PHP source code there is a function `token_get_all`[1] that creates a list of tokens using the lexical scanner of PHP interpreter. I have written a simple parser that uses the tokens, determines the context of the reference and correctly prefixes it. The simple parser was inspired by old Nette build tools that are deprecated now but served as a good start.

The reference can be in many contexts. The type can be inherited, implemented, used as a trait or imported and aliased. It can be in argument typehints or return typehints and more. But the most challenging is to determine if a string contains a reference to the class. Since PHP 5.5 every reference to a class should be written using the special `::class` constant that returns the name of the type and it is defined on all classes by PHP. But this is not the case with old libraries and the preprocessor must support this.

1. https://secure.php.net/manual/en/function.token-get-all.php

It is solved by matching the string to all types defined in dependencies. Since PHP is interpreted and not compiled, Composer provides class loading mechanism for its runtime. Part of that is a class map that Composer generates when it installs the dependencies. It is called class map, but it contains all defined types including interfaces and traits. Because we are prefixing only the dependencies, we can filter the strings using this class map to determine if the string contains a type reference and has to be also prefixed.

I wrote the first prototype of fprochazka/phpstan-compiler[2] using this approach and verified that PHPStan is still working after having its source code preprocessed.

Same logic is applied to NEON configuration files that PHPStan uses for internal configuration of it services.

5.1.2 Preprocessing the extensions

PHPStan supports writing extensions and the source code of the extensions references types in libraries that they check. For example, phpstan/phpstan-nette[3] provides additional context to PHPStan that cannot be easily detected from normal control flow of the application source code. The extension must be able to check if it should handle the analyzed type. The checked type would never match, and the extension would not work correctly if the processor would also prefix the type that the extension is checking.

The PHPStan compiler handles this by preprocessing the extensions separately from other dependencies with different settings. The compiler knows what packages the extension is written for and forbids it from prefixing their class references.

5.1.3 PHAR archiver

Thanks to good native tools in PHP, creating the PHAR archive is simple, and Composer internals was the inspiration for the code that handles it.

2. https://github.com/fprochazka/phpstan-compiler
3. https://github.com/phpstan/phpstan-nette

5.1.4 PHPStan shim package

The compiled PHAR has to be made available somewhere online where it can be downloaded from by the CI server and used in Kdyby. Ondřej created phpstan/phpstan-shim[4] repository for publishing the compiled PHAR, and now it can be installed using Composer.

5.2 Designing new Kdyby Coding Standard

The new Kdyby Coding Standard (KCS) is be based on Consistence Coding Standard (CCS) [21] with slevomat/coding-standard[5] rules.

The standard defines two rulesets `ruleset-5.6` and `ruleset-7.1`. The `ruleset-5.6` is based on `ruleset-7.1` and only removes required PHP 7.1 rules. There will be no standard defined for older PHP versions, and therefore the standard can be applied only on packages that already support only PHP 5.6 and newer.

5.2.1 Nette\Tester support

Tests written in Nette\Tester have `.phpt` extension and not the standard `.php`. KCS will override the defaults and force PHP_CodeSniffer to check both extensions. The tests also execute themselves which creates a side effect. CCS forbids side effects which mean the PHP file must either define a type or execute some operations but never both. This rule is ignored for `.phpt` files.

Writing the tests often requires handling some extreme cases where strictly enforcing the coding standard would be a problem or merely an inconvenience. The Heredoc[6] syntax is allowed in tests for defining larger chunks of strings for comparison.

5.2.2 Exceptions in one file

Nette Coding Standard (NCS) [22] allows having `exceptions.php` file that contains all exception classes for that given namespace. More

4. https://github.com/phpstan/phpstan-shim
5. https://github.com/slevomat/coding-standard
6. https://secure.php.net/manual/en/language.types.string.php#language.types.string.syntax.heredoc

specifically it does not forbid it, and Nette itself is written in that way. Since Kdyby is originally based on NCS, it inherited this practice, and the new standard has an exception for defining multiple types in one file when it is named `exceptions.php` and contains only types whose name ends with `Exception`. A new rule was written to check that the namespace in the `exceptions.php` file adheres to the namespaces definition in Composer autoloading of that package.

5.2.3 Keywords vs constants

CCS defines the constants `TRUE`, `FALSE` and `NULL` as keywords and they must be written in lowercase. But the PHP documentation defines both `NULL` [23] and `TRUE`, `FALSE` [24] as constants. This creates a collision in the standard with a rule that says all constants must be written in uppercase [21]. KCS solves this by requiring `TRUE`, `FALSE` and `NULL` to be written in uppercase.

5.2.4 Short scalar types in PHPDoc

In PHPDoc `int` and `integer` are equivalent, but having two ways to write one thing creates inconsistencies in code. CCS requires the PHPDoc to contain the long variants notation if there is more than one variant available [21]. But this is inconsistent with PHP that does not allow long variants in return types and argument typehints. KCS solves it by requiring that the short notation is used.

5.3 Refactoring summary

Nette 2.3 and 2.4 compatible versions were released for these packages: *Kdyby\Doctrine*, *Kdyby\Console*, *Kdyby\Annotations*, *Kdyby\DoctrineCache*, *Kdyby\Events*, *Kdyby\Translation*, *Kdyby\RequestStack*, *Kdyby\Monolog*, *Kdyby\Clock*.

PHPStan is automatically checking these packages: *Kdyby\Doctrine*, *Kdyby\Console*, *Kdyby\Annotations*, *Kdyby\DoctrineCache*, *Kdyby\Events*, *Kdyby\Translation*, *Kdyby\CodingStandard*, *Kdyby\StrictObjects*, *Kdyby\RequestStack*, *Kdyby\Monolog*, *Kdyby\Clock*, *Kdyby\GeocoderSeznamMaps*, *Kdyby\GeocoderLogging*, *Kdyby\GeocoderGoogleMaps*-

Proxied, *Kdyby\GeocoderBestMatch*, *Kdyby\DoctrineMagicAccessors*, *Kdyby\DoctrineCollectionsReadonly*.

New Kdyby Coding Standard is applied and enforced on these packages: *Kdyby\Console*, *Kdyby\Annotations*, *Kdyby\DoctrineCache*, *Kdyby\Events*, *Kdyby\Translation*, *Kdyby\CodingStandard*, *Kdyby\StrictObjects*, *Kdyby\RequestStack*, *Kdyby\Monolog*, *Kdyby\Clock*, *Kdyby\GeocoderSeznamMaps*, *Kdyby\GeocoderLogging*, *Kdyby\GeocoderGoogleMapsProxied*, *Kdyby\GeocoderBestMatch*, *Kdyby\DoctrineMagicAccessors*, *Kdyby\DoctrineCollectionsReadonly*.

6 Conclusion and Future work

The primary goal of this thesis was to release as many new versions as possible of Kdyby packages that are compatible with new versions of libraries they integrate with. The goal to start checking the packages with PHPStan and have a coding standard is important but not critical. And finally separating the packages into smaller packages with fewer responsibilities has advantages but the biggest one is marketing where the programmers will not feel intimidated by the amount of code in the package and will be more inclined to actually install it and therefore is also not as critical.

Almost all of the most popular packages have now versions that support current Nette, Doctrine, and Symfony. And only a few of those do not have PHPStan, or Kdyby Coding Standard configured on CI server. Not all packages are resolved, but that is fine. If the work had been rushed, it would beat the purpose of refactoring them.

The amount of work covered by this thesis represents several months of work but only a fraction of what have already been invested into Kdyby by the maintainers and community.

The roadmap chapter 4 of this thesis covers the big architectural changes for Kdyby packages and only a part of that plan was implemented. By solving them, many of the open issues will be also resolved. As of writing this, there are 188 open issues and pull requests that have to be solved or reviewed.

The issues and pull requests alone are several months of work and the architectural changes will take at least half a year to implement.

Bibliography

[1] The PHP Group. *PHP Manual - What is PHP?* 2017. URL: https://secure.php.net/manual/en/intro-whatis.php.

[2] Free Software Foundation. *What is free software and why is it so important for society?* 2017. URL: https://www.fsf.org/about/what-is-free-software.

[3] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: https://martinfowler.com/articles/injection.html.

[4] Gregor Kiczales et al. "Aspect-oriented programming". In: *ECOOP'97—Object-oriented programming* (1997), pp. 220–242.

[5] John Vlissides et al. "Design patterns: Elements of reusable object-oriented software". In: *Reading: Addison-Wesley* 49.120 (1995), p. 11.

[6] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014. ISBN: 1484200772.

[7] Martin Fowler and Matthew Foemmel. "Continuous integration". In: *Thought-Works) http://www.thoughtworks.com/ContinuousIntegration.pdf* (2006), p. 122.

[8] Nette Foundation. *Introduction to Nette Framework*. 2017. URL: https://doc.nette.org/en/2.4/getting-started.

[9] Composer Community. *Introduction - Composer*. 2017. URL: https://getcomposer.org/doc/00-intro.md.

[10] Ryan Boyd. *Getting started with OAuth 2.0*. " O'Reilly Media, Inc.", 2012.

[11] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[12] SensioLabs. *What is Symfony*. 2017. URL: http://symfony.com/what-is-symfony.

[13] Inc Pivotal Software. *What can RabbitMQ do for you?* 2017. URL: https://www.rabbitmq.com/features.html.

[14] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. " O'Reilly Media, Inc.", 2015.

[15] Salvatore Sanfilippo. *Redis*. 2017. URL: https://redis.io/.

[16] Ondřej Mirtes. *PHPStan - PHP Static Analysis Tool*. 2017. URL: https://github.com/phpstan/phpstan.

[17] PHP Framework Interop Group. *PHP Standards Recommendations*. 2017. URL: http://www.php-fig.org/psr/.

[18] Jordi Boggiano. *Monolog - Logging for PHP*. 2017. URL: https://seldaek.github.io/monolog/.

[19] David Grudl. *Nette Tester – enjoyable unit testing*. 2017. URL: https://tester.nette.org/en/.

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

[21] Vašek Purchart. *Consistence Coding Standard*. 2017. URL: https://github.com/consistence/coding-standard/blob/cb0d087e12256c9b3fa6901ca5ae9a4d0e0cb3bd/consistence-coding-standard.md.

[22] Nette Foundation. *Nette Coding Standard*. 2017. URL: https://nette.org/en/coding-standard.

[23] The PHP Group. *PHP Manual - Language Reference - Types - NULL*. 2017. URL: https://secure.php.net/manual/en/language.types.null.php.

[24] The PHP Group. *PHP Manual - Language Reference - Types - Booleans*. 2017. URL: https://secure.php.net/manual/en/language.types.boolean.php.