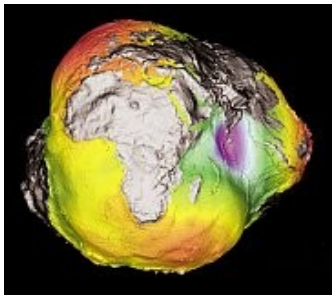
Fourth VuFind user meeting in Constance, Germany, October 7th, 2015

“Have we become a little chubby around the waist?”



The “Potsdam potato” http://www.spektrum.de/news/um-die-hueften-fuelliger/600896

Thoughts and proposals for more components and flexibility in the VuFind 2/3 architecture based on principles of Zend Framework 3

swissbib

mandated by swissuniversities

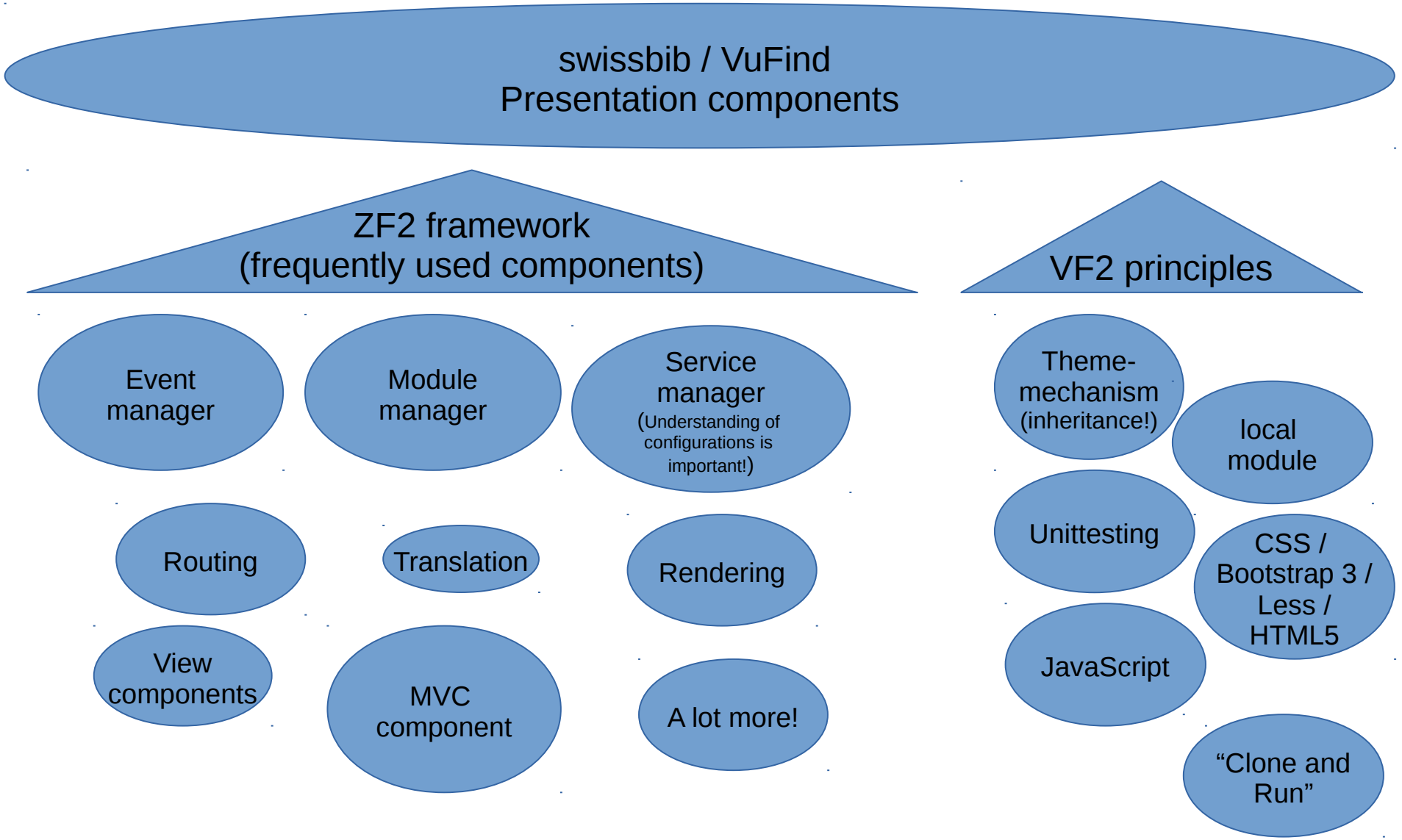
Günter Hipler, swissbib project, University of Basel Library

Overview

I. Short summary of the current VuFind2 / ZF2 architecture

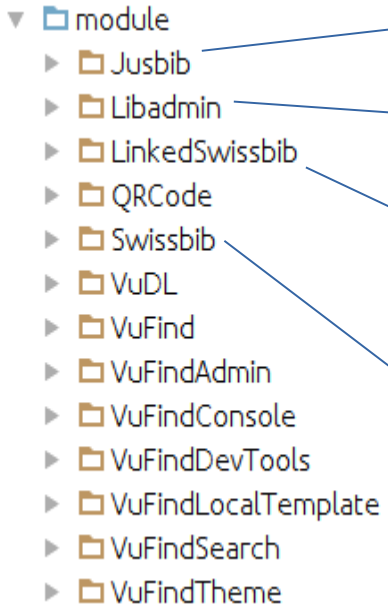
II. Experience gathered during the last two years:
What were we able to implement so far?
Where did we come up against certain limits?

III. New possibilities for the VuFind 3 architecture based on principles of ZF3 and PHP7


swissbib / VuFind
Presentation components
ZF2 framework
(frequently used components)
VF2 principles
Event manager
Module manager
Service manager
(Understanding of configurations is important!)
Routing
Translation
Rendering
View components
MVC component
A lot more!
Theme-mechanism (inheritance!)
local module
Unittesting
CSS / Bootstrap 3 / Less / HTML5
JavaScript
“Clone and Run”

What were we able to implement in swissbib so far, based on the current architecture?

I. Implementation of additional modules



Portal for legal aspects
(https://jus.swissbib.ch)

Administration of information for more than 900 institutions involved in swissbib (http://admin.swissbib.ch/libadmintest)

Integration of linked data produced by linked.swissbib.ch within our presentation component based on ElasticSearch (https://github.com/linked-swissbib)

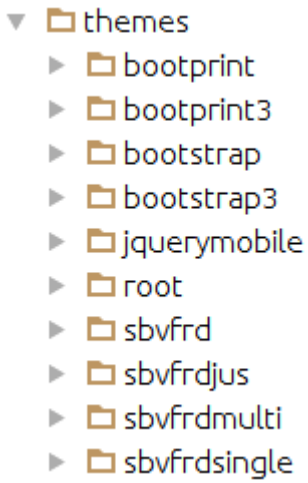
Enhancements for the national catalog system www.swissbib.ch which includes http://baselbern.swissbib – a full 'user catalog' with Aleph connection.
(see https://github.com/swissbib/vufind)

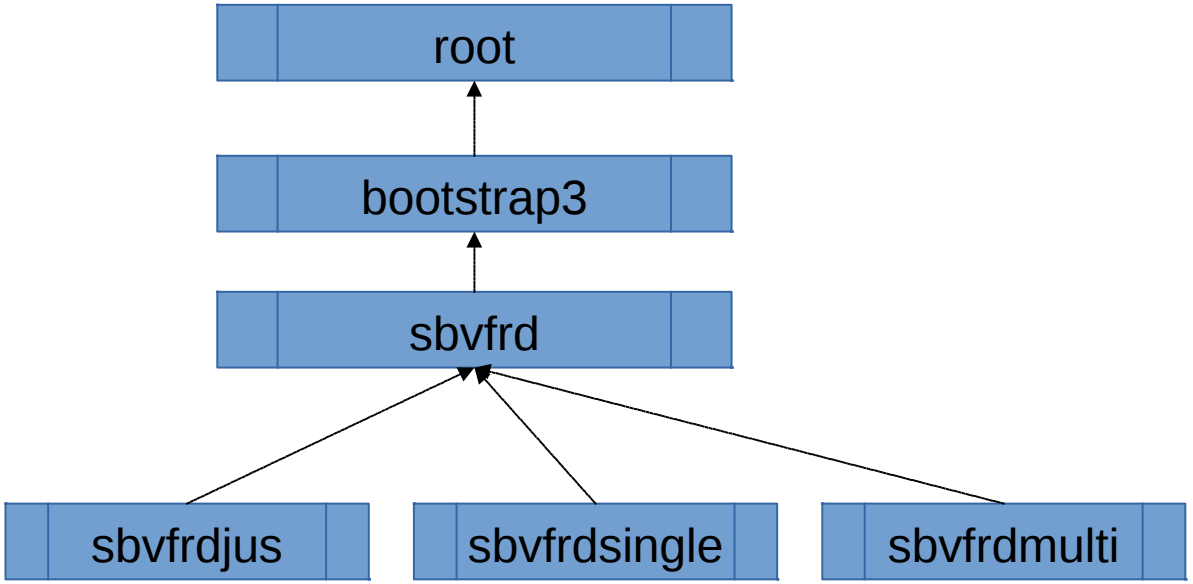
What were we able to implement in swissbib so far, based on the current architecture?

II. Extensive interface enhancements

a) Until May 2015, we used our own unique interface (resulting in very good recognition of the swissbib service, but requiring considerable efforts by the swissbib team)

b) Afterwards, we changed to Responsive Design implementation of VF2 with partial extensions by swissbib





Short summary of the architecture overview:

→ the VuFind application is based on the extensive ZF2 framework, particularly on the Model / View / Controller (MVC) Workflow

→ MVC is based on a ‘multiplicity’ of components. The interaction of these can be changed by the application. VuFind uses these possibilities ingeniously, for example within the theme implementation.

Disadvantages of MVC in the existing form:

→ it is heavyweight (the flow for processing a REQUEST to get a RESPONSE is very costly and resource intensive).

→ it is not the best alternative for every service request.
(Examples where more lightweight processing is desirable: cover creation, autosuggestion ...)

Limitations of the current VF2 architecture

I. Use of “application modules” instead of “components”.

II. All of the VuFind (application) modules plus all of the dependent components are part of a single GitHub repository. The “Clone and Run” principle allows institutions with fewer technical resources a quicker start using the VuFind application.

III. Use of the heavyweight MVC architecture as the only principle for every kind of functionality provided by VuFind

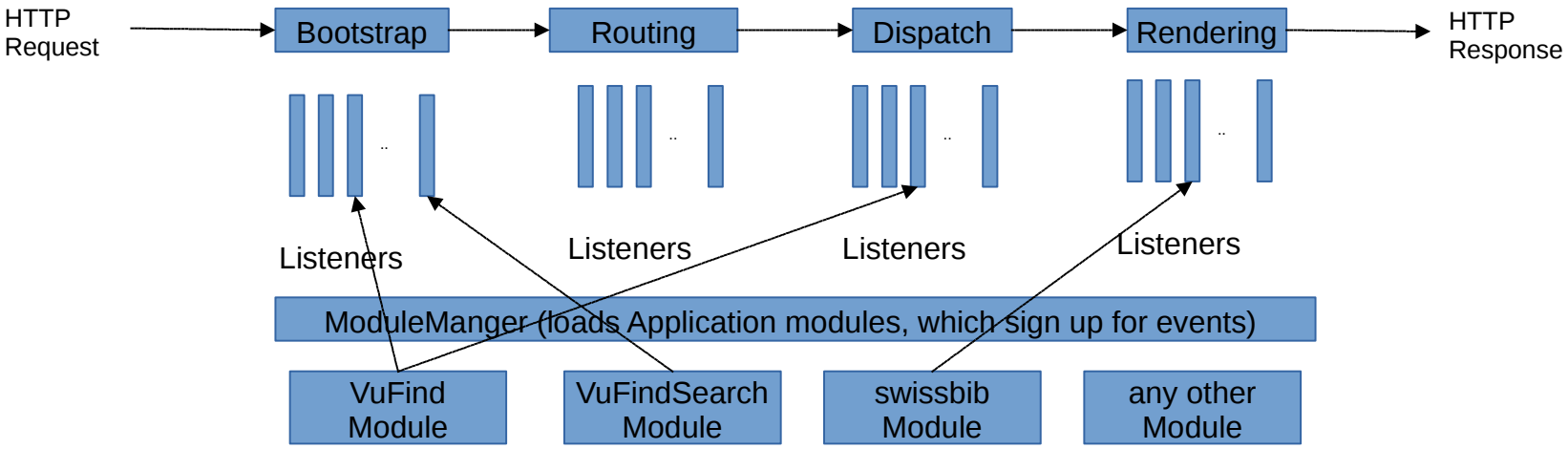
Problem areas:

– lack of flexibility
– resource intensive, suboptimal implementation of single use cases.
– It reminds one a little bit of our “chubby” integrated library systems.

“**Application module**” versus “**component**”

My definition for an ‘application module’ in the VuFind context:
“Code provided by ZF2 Module Manager to be part of the MVC workflows”

Schema of ZF2 “Model View Controller” workflows:
see. http://www.zend.com/en/webinars/recorded/show/340_the%20mvc%20architecture%20of%20zf2



Even if it is possible to install modules as “libraries” using composer, they have to be loaded with the ModuleManager which couples them closely to MVC and makes it difficult to use them with other lightweight workflows.

This makes it much more difficult to use single aspects or functionality like interfaces and base classes for search outside of the original VuFind Application context.

swissbib

Where did we come up against limits?

More quotes und comments for a better distinction between **“application module”** and **“component”**

Evan Coury (implementor of ModuleManager): http://www.zend.com/en/webinars/recorded/show/341_introducing%20the%20zend%20framework%202%20modulemanager “Resuable pieces of functionality that can be used to construct a more complex application”
“ModuleManager – It's the component that will consume Module classes”
“What can Modules be? Anything! Plugins, Themes, Libraries, Applications”

Enrico Zimuel: http://www.zend.com/en/webinars/recorded/show/340_the%20mvc%20architecture%20of%20zf2
“**Resuable pieces of functionality** that can be used to construct a more complex application”
“A module is all related code and assets that solve a specific problem. **Modules inform the MVC** about services and event listeners.”

Mike Willbanks : https://speakerdeck.com/mwillbanks/zf2-writing-service-components
Modules are more specifically for ZF2 Applications
ServiceComponents [GH.: libraries installed via Composer] are resusable libraries for any code base
BaseRule:
If it involves the MVC; it should more than likely be a module.

Until now the only global VuFind Component (**vufind-org/vufindhttp**):
https://packagist.org/packages/vufind-org/vufindhttp
Just at the beginning a local module, it was refactored as a component with a dependency in Composer.

Introduction

VuFindHttp contains a convenience class built around Zend\Http\Client which allows creation of proxy-aware clients and convenient GET/POST behavior. This is part of the VuFind project (http://vufind.org) but may be useful for any software dealing with HTTP calls.

Installation

The recommended method for incorporating this library into your project is to use Composer (http://getcomposer.org).

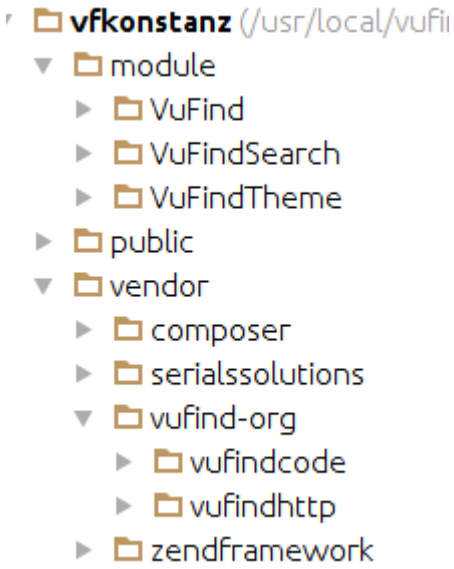
swissbib

Where did we come up against limits?

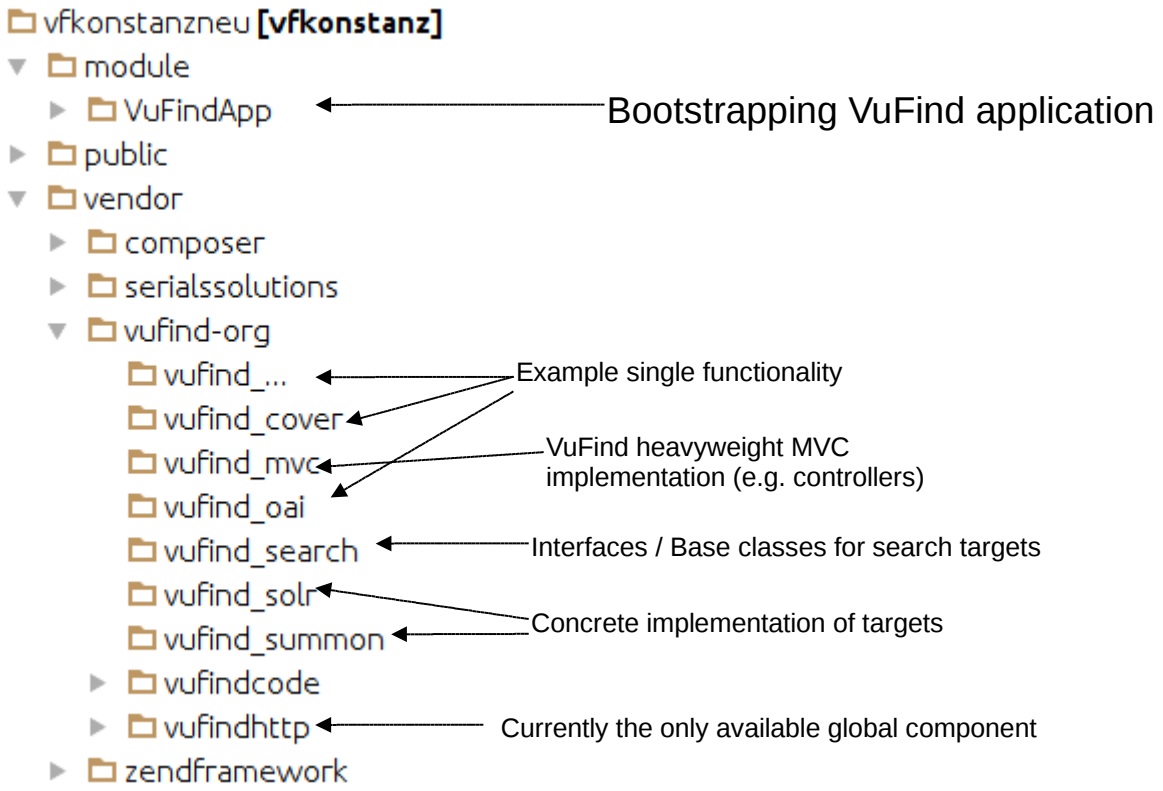
A proposal how to split up the main VuFind modules into single global components (libraries)

More global components (integration with composer) – fewer local modules

Current partitioning



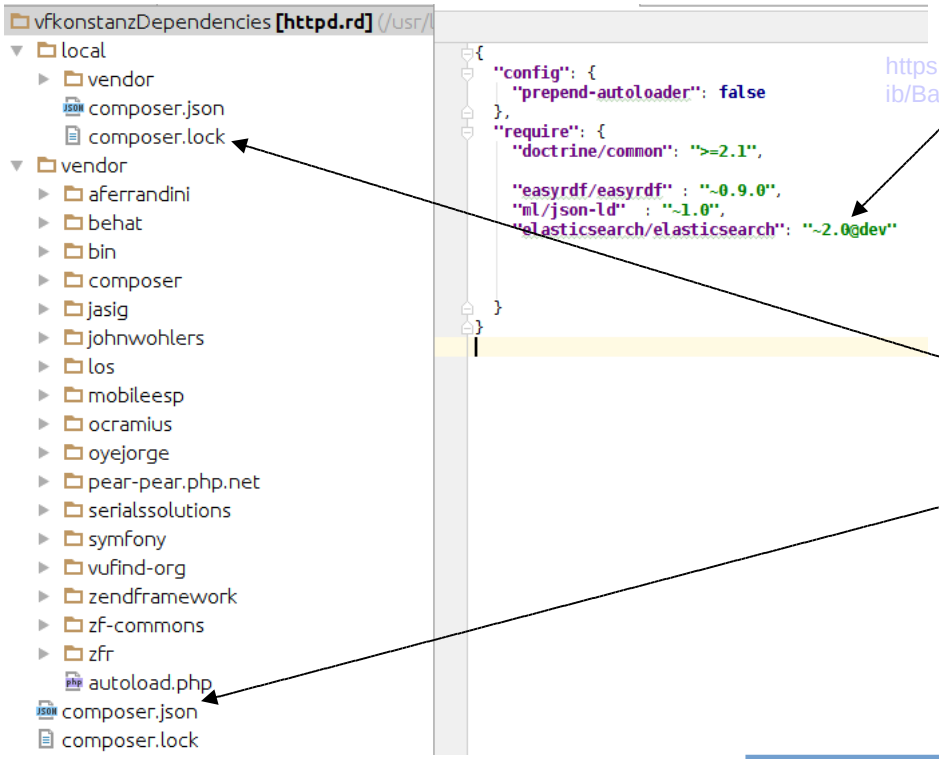
Proposal for global components



swissbib

Where did we come up against limits?

Requirement: additional dependencies



https://github.com/linked-swissbib/vufind/tree/vfsb/linked/module/LinkedSwissbib/src/LinkedSwissbib/Backend/Elasticsearch

problem: use of additional libraries which are not already provided by the VuFind repository
mailing list thread: [VuFind-Tech] Local composer dependencies
http://sourceforge.net/p/vufind/mailman/message/34334936/

Our current solution: local dependencies in parallel to VuFind composer.json to avoid merge conflicts (we are not very happy with this solution)

My proposal for VuFind3:

– do not use “Clone and Run” principle in the future
– additional dependencies should not be part of the VuFind repository.

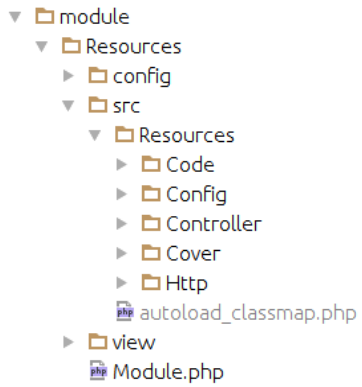
Perhaps as a compromise: creation of an ‘additional’ complete branch (containing all the dependencies) only for a productive release

swissbib

Where did we come up against limits?

Current swissbib solution for cover creation:

resources.swissbib.ch: reduced MVC workflow on dedicated hosts



'every' single request still uses a (simplified) MVC workflow on dedicated hosts :
– Loading of only one module
– Load of configuration (less important because of caching in production)
– Routing
– Bootstrapping of one module and registration of events only for this
– Controller Dispatching
– Rendering (within cover context only images)

Our current solution (primarily to reduce requests on hosts):
– "Outsourcing" **of cover creation** (still local module) on dedicated hosts with an adjusted **MVC** workflow

https://resources.swissbib.ch/Cover/Show?isn=0133994619&size=small

– **Disadvantages**:
→ **Duplication of code** because of the lack of global components in VuFind
→ still **ZF2/MVC** – currently we do not have an alternative
→ less functionality (no special theme rendering of images, which is not important for us)

Summary:
For specific and simple requests like creation of covers or autosuggestion – which take place very often – the classic MVC workflow is not necessary and is not the best choice because of the high server load and slower user experience (longer response times)

Wanted: 'lightweight workflows'

Light MVC in swissbib for cover creation

Summary:
Advantages of global components and individual installation of dependencies via composer

1) More flexible usage of single components
   (by VuFind itself for user interface and/or API functionality as well as third services outside VuFind)

2) Merge conflicts will be easier avoided

3) Single global repositories makes it easier to divide the responsibility for the implementation of special functionalities among various institutions. (Villanova, Finna, Finc, BSZ, swissbib …..)

4) Easier versioning: not all the components change with the same frequency. Perhaps releases could be deployed more often.

First outlook for changes in the upcoming ZF3

1. Partitioning of only one ZF2 Repository into many single repositories which makes it easier to use them as components.

2. Implementation of the PHP Specification Request 7 (PSR7) within the Zend Framework as the basis for new so-called Middleware-Container

3. Refactoring of central components part of the framework

   (especially ServiceManager and EventManager)

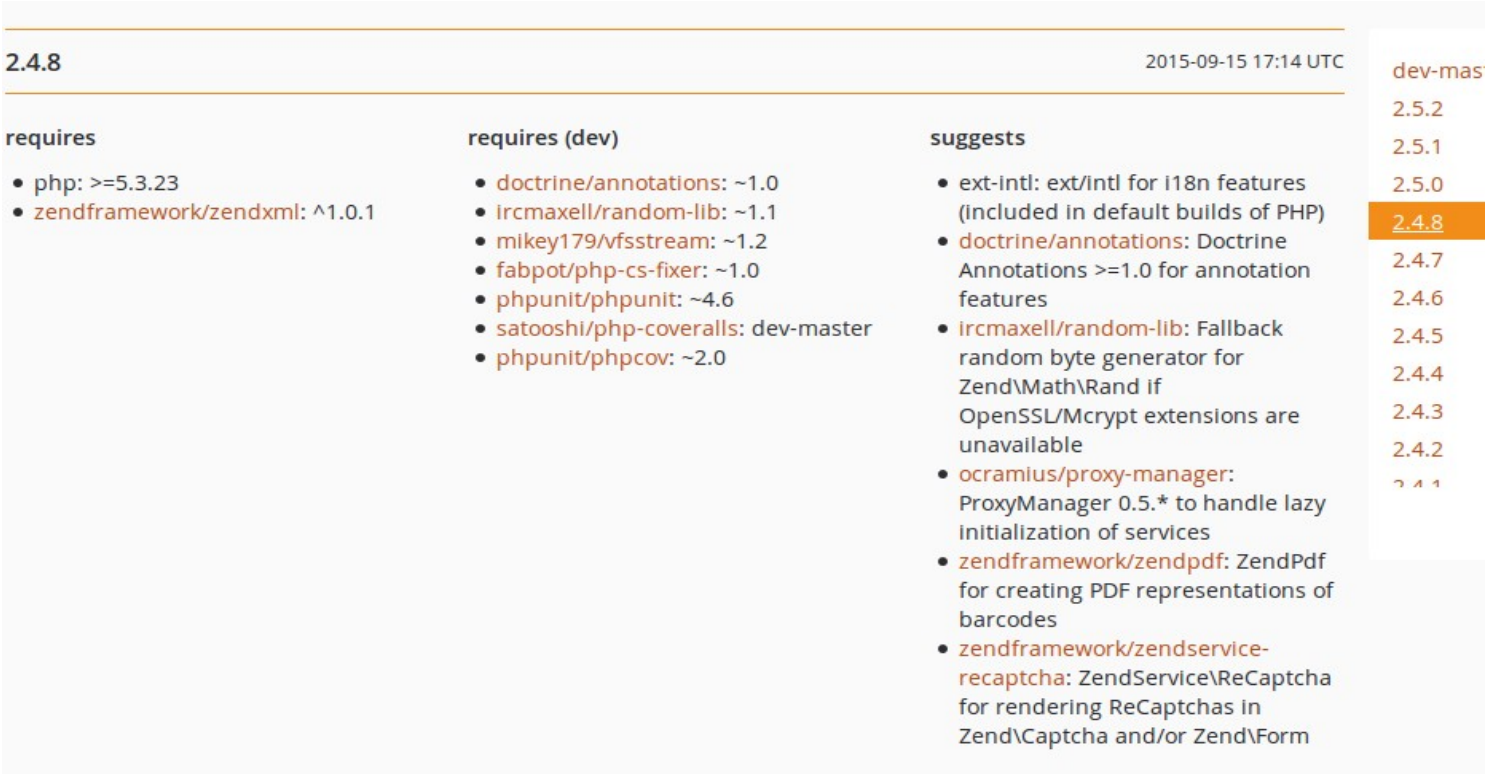
   https://github.com/zendframework/zend-eventmanager/pull/4

   https://twitter.com/mwop/status/643545615478226945

Partitioning of only one repository into many single components

Version < 2.5 of Framework in only one repository
(which is used by the current VuFind2)

https://packagist.org/packages/zendframework/zendframework



2.4.8 2015-09-15 17:14 UTC

requires
- php: >=5.3.23
- zendframework/zendxml: ^1.0.1

requires (dev)
- doctrine/annotations: ~1.0
- ircmaxell/random-lib: ~1.1
- mikey179/vfsstream: ~1.2
- fabpot/php-cs-fixer: ~1.0
- phpunit/phpunit: ~4.6
- satooshi/php-coveralls: dev-master
- phpunit/phpcov: ~2.0

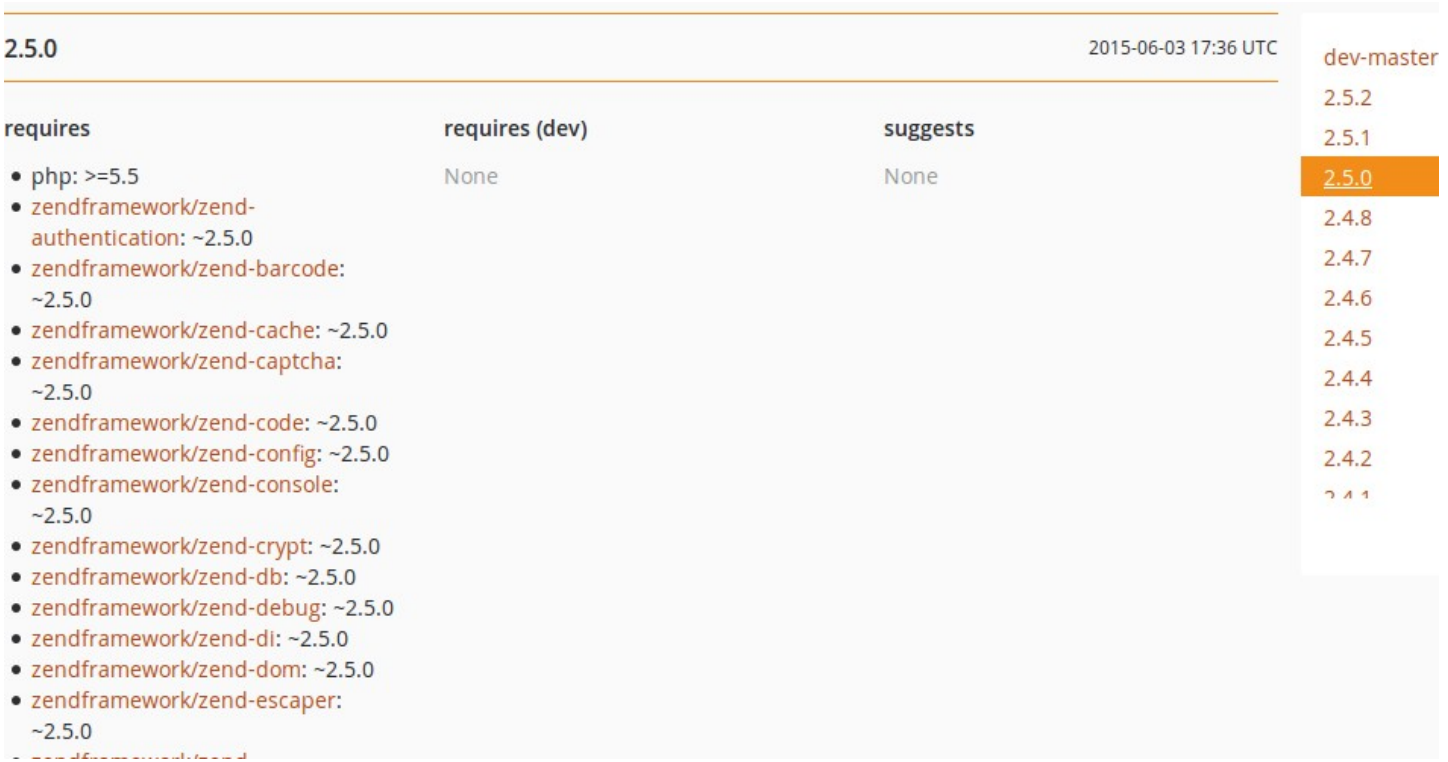
suggests
- ext-intl: ext/intl for i18n features (included in default builds of PHP)
- doctrine/annotations: Doctrine Annotations >=1.0 for annotation features
- ircmaxell/random-lib: Fallback random byte generator for Zend\Math\Rand if OpenSSL/Mcrypt extensions are unavailable
- ocramius/proxy-manager: ProxyManager 0.5.* to handle lazy initialization of services
- zendframework/zendpdf: ZendPdf for creating PDF representations of barcodes
- zendframework/zendservice-recaptcha: ZendService\ReCaptcha for rendering ReCaptchas in Zend\Captcha and/or Zend\Form

dev-mas
2.5.2
2.5.1
2.5.0
2.4.8
2.4.7
2.4.6
2.4.5
2.4.4
2.4.3
2.4.2

Why split them at all?
"But you can already install components individually!"
True, but if you knew how that occurs, you'd cringe
https://mwop.net/blog/2015-05-15-splitting-components-with-git.html

ZF2 construction of the repository for versions >= 2.5 (as preparation for ZF3)

https://packagist.org/packages/zendframework/zendframework



Version >= 2.5: **around 50 single repositories**

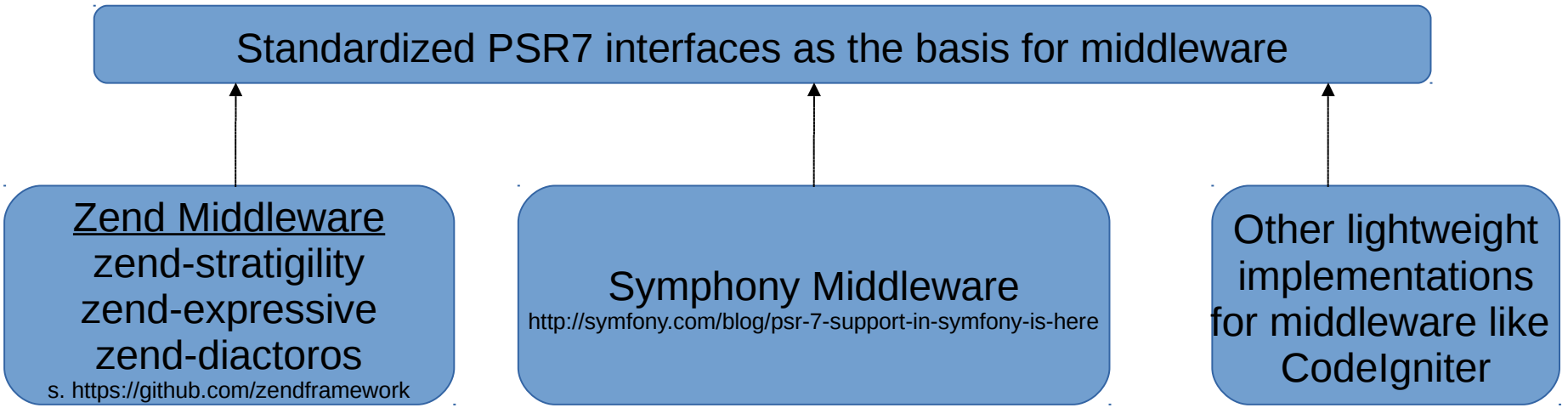
swissbib

New partitioning of the Framework into many single components with one Meta-Repository

2. Implementation of the PHP Specification Request 7 (PSR7) within the Zend Framework

PSR 7 in a nutshell:
unified interfaces to represent HTTP messages and URIs

http://www.php-fig.org/psr/psr-7/
https://mwop.net/blog/2015-01-26-psr-7-by-example.html
https://mwop.net/blog/2015-01-08-on-http-middleware-and-psr-7.html
http://blog.alejandrocelaya.com/2015/09/12/my-first-approach-to-zend-expressive/



Unified interfaces make it a lot easier to combine and to use different approaches for more alternatives and flexibility within VuFind, and enable other services to use VuFind functionality as a whole and/or in part

Is the transition to the new version of Zend as complicated as the change was from VuFind1 to VuFind2?

→ the refactoring of the central components are practically 'free of charge'
(with the exception of minor interface adaptations)

→ we hardly have to change anything

→ in my opinion, *for VuFind3 we should use the opportunity of a refactoring* based on the mentioned enhancements (middleware as well as single components, instead of only application modules) to eliminate the known weaknesses, as well as to gain additional possibilities for enhancements and increased flexibility in using VuFind.

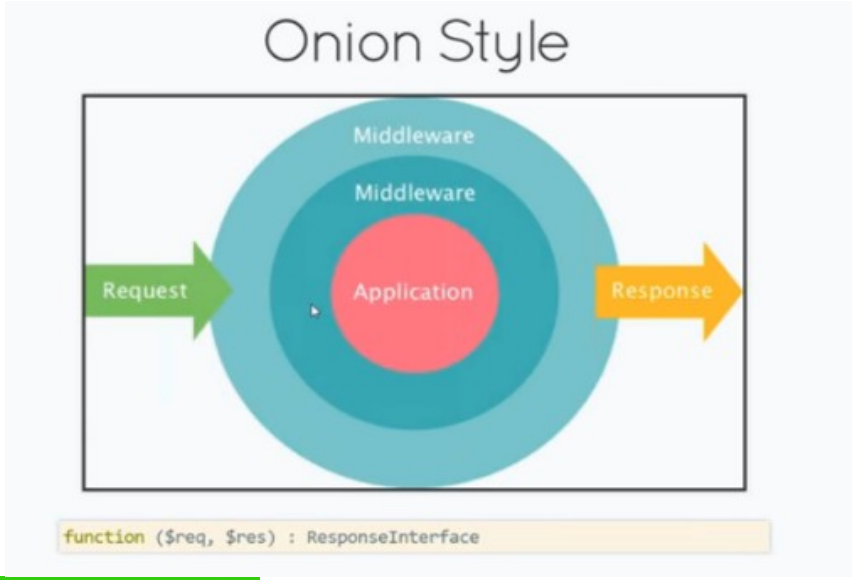
Middleware in a nutshell

– Middleware as a frame (container) for processing the REQUEST into a RESPONSE

– it is possible to combine different middleware containers like shells

– every shell uses the unified interfaces of PSR7 to exchange data from the REQUEST into the RESPONSE

– thanks to unified interfaces, containers can take over different tasks and can be replaced by alternative implementations



http://www.zend.com/en/webinars/recorded/show/3033_stratigility%20middleware%20for%20php%20and%20psr-7

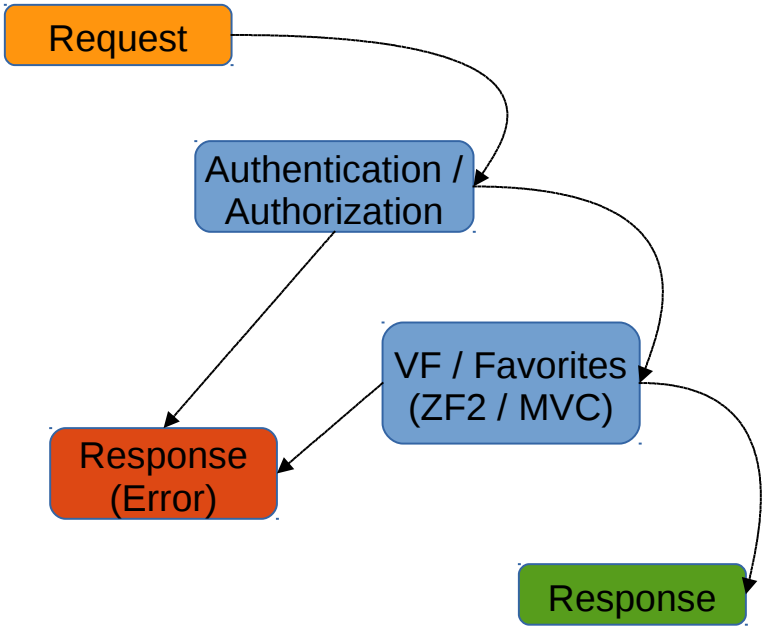
https://www.youtube.com/watch?v=B2YqevRpi6E

swissbib

Diagram for the construction of a middleware stack

Middleware in a nutshell

Composition of middleware shells

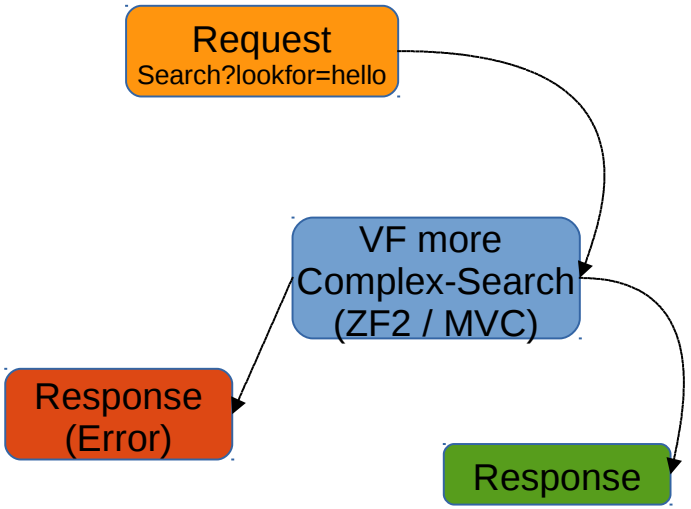


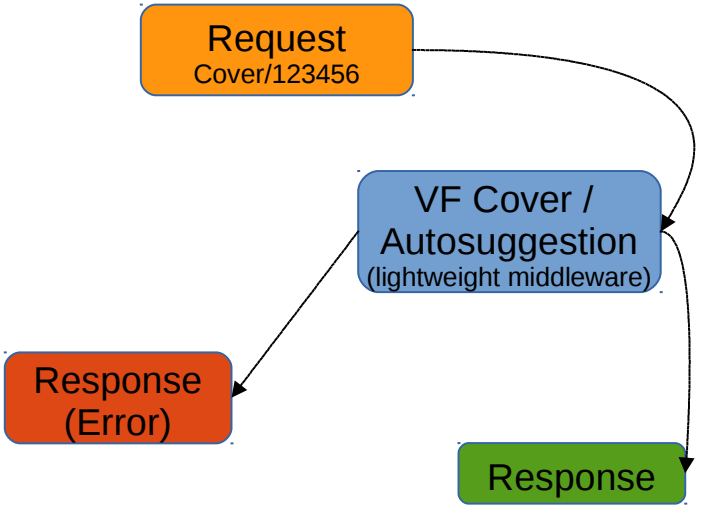
swissbib

Examples of possibilities to assemble middleware containers

Middleware in a nutshell

Using various middleware containers



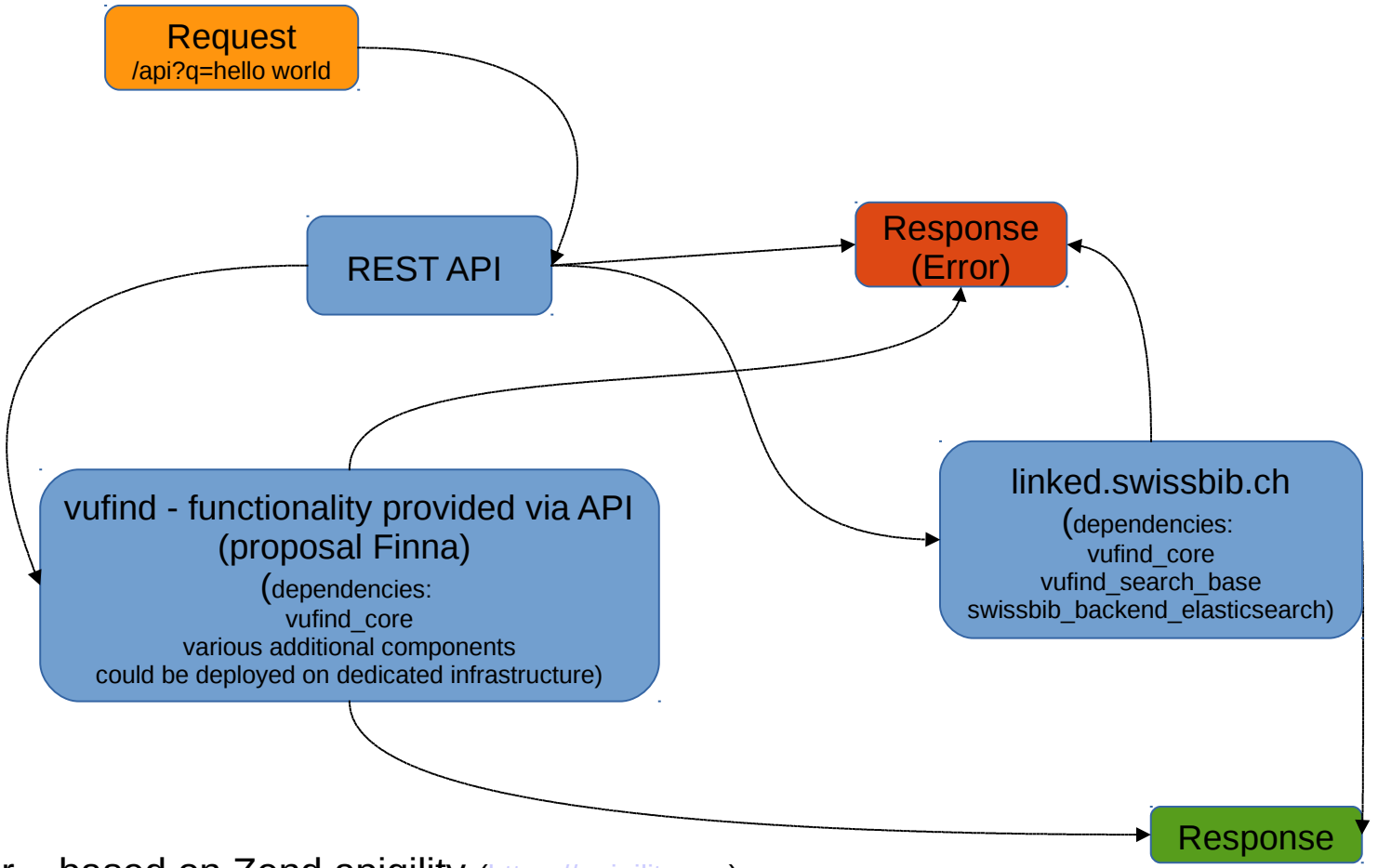


“heavyweight” ZF2 / MVC implementation

“lightweight” middleware for image requests or autosuggestion

Middleware in a nutshell

Using alternative middleware



– REST API Container – based on Zend apigility (https://apigility.org)
– selected single repositories implement the functionality

swissbib

Examples of possibilities to assemble middleware containers

What does middleware in code look like?

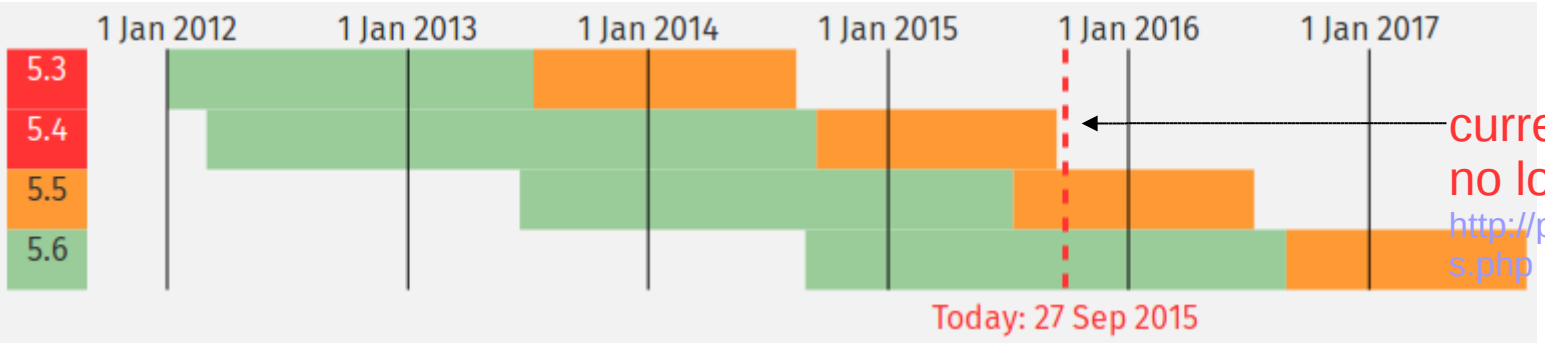
More code examples
https://github.com/zendframework/zend-expressive
http://blog.alejandrocelaya.com/2015/09/12/my-first-approach-to-zend-expressive/

```php
<?php
    use Zend\Stratigility\MiddlewarePipe;
    use Zend\Expressive\AppFactory;
    require __DIR__ . '/../vendor/autoload.php';


    /* lighweight middleware for cover requests */
    $cover_lightweight_vufind_cover_repository_Middleware = new MiddlewarePipe();

    /* middleware implementing the apigility REST framework */
    $apigility_REST_vufind_rest_respository_Middleware = new MiddlewarePipe();

    /* very specific micro framework for what ever */
    $very_specific_Middleware = new MiddlewarePipe();

    $app = AppFactory::create();

    /* use the standard MVC middleware implementing the classic MVC flow -
    what we have by now in VuFind 2*/
    $app->pipe('/');
    $app->pipe('/cover', $cover_lightweight_vufind_cover_repository_Middleware);
    $app->pipe('/api', $apigility_REST_vufind_rest_respository_Middleware);
    $app->pipe('/specific', $very_specific_Middleware);

    $app->run();
```

swissbib

Examples of simple middleware code

Overview of the current PHP versions



current VuFind Version
no longer supported!
http://php.net/supported-versions.php

Short notes about **PHP 7** (**productive** release in October **2015**) :
Zend Engine was pressured by Facebook's HHVM development
→ **fundamental changes and reworking**
→ large improvements in performance
→ TypeHints
→ ZF3 minimum: PHP 5.5 / will be tested against PHP7
https://www.zend.com/en/resources/php7_infographic

My proposal for the requirements of PHP versions in VuFind:
VF2 → Change to PHP 5.5 / PHP 5.6 with version 2.6 latest 2.7
VF3 → development should be tested against PHP 7
at the end of 2016: at least a recommendation to use PHP 7

Possible next steps?

→ VuFind Summit: October 2015
Start of discussions about the VuFind 3 roadmap
(as many users as possible should take part in the discussion – one reason for this presentation)

→ Discussion should not take place only at the VuFind summit. All 'active coders' should begin to familiarize themselves with the new additional possibilities of the latest PHP and ZF developments. This would help a lot to develop an improved design and architecture for VuFind3 as a community.

→ We do not have to wait until ZF3 is finally released. All the refactored central components are already available. There exists implementation for the new middleware containers and PSR7 as well as the partition into single repositories since version 2.5.

https://github.com/zendframework/zend-stratigility
https://github.com/zendframework/zend-diactoros
https://github.com/zendframework/zend-expressive
https://apigility.org/ (for interfaces based on REST principles)

Discussion / questions / cries of boo / enthusiasm ??

Links for more information about the swissbib projects and the regulatory framework

swissbib

Blog:	http://swissbib.blogspot.ch/	
Twitter:	www.twitter.com/swissbib	
Project Wiki:	www.swissbib.org	(write short mail to get member status)
Code swissbib:	https://github.com/swissbib	(swissbib „classic“)
Code linked-swissbib:	https://github.com/linked-swissbib	
Project mail:	swissbib-ub@unibas.ch	

swissuniversities

SUK-Programm 2013-2016 P-2 «Wissenschaftliche Information: Zugang, Verarbeitung und Speicherung»

http://www.swissuniversities.ch/en/organisation/projekte-und-programme/suk-p-2-wissensch-information-zugang-verarbeitung-speicherung/

Personal coordinates:
guenter.hipler@unibas.ch
Tel.: +41 61 267 31 12