Belangrijke begrippen

CMS / Content Management System
Een webapplicatie, die een eindgebruiker een op een gebruikersvriendelijke manier het beheer van inhoud ter beschikking stelt

PHP
Een heel wijd verspreide populaire programeertaal voor webapplicaties.

Framework (PHP)
Een toolbox met oplossingen voor vaak voorkomende probleemstellingen voor webapplicaties.

Coding Standard
Een gestandaardiseerde manier om broncode te structureren. Bovendien wordt in een coding standard bijvoorbeeld een bepaalde benaming voor variabelen vereist. Door een coding standard wordt de samenwerking van meerdere ontwikkelaars van code eenvoudiger, omdat de broncode bij iedereen dezelfde structuur heeft.

Interface
Het woord interface heeft meerdere betekenissen. Ten eerste betekent Interface de visuele weergave op een scherm voor een eindgebruiker. Ten tweede wordt hiermee een concept van object georiënteerd programmeren bedoeld. Interfaces definiëren een formaat / een structuur, aan die componenten moeten voldoen.

Unit Test
Unit Tests zijn applicaties, die de hoofdapplicatie testen. In plaats van een mens, die een applicatie daadwerkelijk test kan ook een unit test geschreven worden, welk onderdelen van een systeem valideert.

Third Party Components
Componenten van een andere partij ter beschikking gesteld worden

Dependency Injection (DI)
Een concept om objectstructuren binnen een object georiënteerde applicatie te beheren. Dependency Injection minimaliseert de afhankelijkheden binnen de applicatie en verhoogt de flexibiliteit.

IDE
Een IDE is applicatie om broncode te schrijven. Feitelijk is een IDE te vergelijken met een teksteditor, die functies ter beschikking stelt die het programmeren vereenvoudigen.

Literatuurlijst

Overzicht Frameworks	http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#PHP
Zend Framework documentatie	http://framework.zend.com/docs/overview
SymfonyTutorial – Building a blog	Creating a blog in Symfony2 — symblog - A Symfony2 Tutorial
Algemene vergelijking – YII, Zend	http://www.sheldmandu.com/php/php-mvc-frameworks/yii-vs-zend-vs-code-igniter-compared
Performance	http://www.yiiframework.com/performance/ http://www.slideshare.net/aditseng/comparing-web-frameworks (vanaf pagina 25)
VimPluginCakePHP	http://www.vim.org/scripts/script.php?script_id=3650
VimPlugin Symfony2	https://github.com/docteurklein/vim-symfony
Eclipse Symfony2 Plugin	http://symfony.dubture.com/
PHPStorm Symfony2	http://www.jetbrains.com/phpstorm/webhelp/enabling-a-command-line-tool.html
Eclipse CakePHP Plugin	http://opencakefile.sourceforge.net/
Netbeans Yii Framework Plugin	http://www.yiiframework.com/wiki/83/netbeans-ide-and-yii-projects/
Eclipse Yii Framework Plugin	http://yiiclipse.maziarz.org/features/
Netbeans Zend Plugin	https://blogs.oracle.com/netbeansphp/entry/using_zend_framework_with_netbeans
Eclipse Zend	http://dionysus.uraganov.net/software/plugins-for-zend-studio-for-eclipse/
Notepad++ Zend Plugin	http://sourceforge.net/projects/zfassistant/
Symfony Algemen	http://en.wikipedia.org/wiki/Symfony
Zend Framework Roadmap (LTS)	http://framework.zend.com/wiki/display/ZFDEV2/Zend+Framework+2.0+Roadmap
PSR-2	https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md
RESTful services	http://www.ibm.com/developerworks/webservices/library/ws-restful/
Coding Standards YII	http://www.yiiframework.com/doc/guide/1.1/en/basics.convention https://github.com/yiisoft/yii/
Coding Standards Symfony2	http://symfony.com/doc/2.0/contributing/code/standards.html http://symfony.com/doc/2.0/contributing/code/conventions.html

Coding Standards CakePHP	http://book.cakephp.org/2.0/en/contributing/cakephp-coding-conventions.html
Coding Standards Zend	http://framework.zend.com/manual/1.12/de/coding-standard.coding-style.html
CakePHP Packages	http://plugins.cakephp.org/ http://stackoverflow.com/questions/1828768/cakephp-shopping-cart http://www.ohloh.net/p/bakesale https://github.com/neilcrookes/CakePHP-Blog-Plugin http://milesj.me/blog/read/changelog-forum-3.1
Zend Packages	https://github.com/vespolina https://github.com/Sylius https://github.com/Herzult/HerzultForumBundle https://github.com/sonata-project/SonataNewsBundle
Analysis various frameworks	https://github.com/WebDevPT/Yii-Blog-Enhanced
Requirements	http://symfony.com/doc/current/reference/requirements.html http://book.cakephp.org/1.3/view/908/Requirements http://www.yiiframework.com/doc/guide/1.1/en/quickstart.installation http://framework.zend.com/manual/1.12/de/requirements.introduction.html
Omvang frameworks	http://framework.zend.com/manual/1.12/de/requirements.introduction.html http://symfony.com/doc/current/components/index.html http://www.yiiframework.com/features/ http://book.cakephp.org/2.0/en/core-libraries.html
Performance	http://www.devshed.com/c/a/PHP/7-PHP-Frameworks-Tested-For-Speed/2/
Features	http://en.wikipedia.org/wiki/Yii http://symfony.com/doc/current/components/index.html http://cakephp.org/pages/features http://framework.zend.com/manual/2.0/en/index.html
Flexibeliteit	http://de.slideshare.net/e.zimuel/a-quick-start-on-zend-framework-2

Bijlage REST API

De backend is via REST API te beheren. Symfony2 biedt in de standaard distributie geen functionaliteit voor het maken van een REST API aan. De REST API wordt door het FOSRestBundle mogelijk gemaakt. Dit is een uitdagende bundle, welke b.v. voor een correcte routing zorgt en een output in een passend formaat genereert (json, xml).

Via Rest verander je principieel één resource per request. Per resource in WM3 kunnen er de volgende operaties zijn:

- GET: Resource opvragen + zinvolle gerelateerde resources
- POST: Een nieuw resource aanmaken
- PUT: Een resource wijzigen
- DELETE: Een resource verwijderen
- EDIT: Formulier voor een bestaande resource opvragen die nodig is voor het updaten van een resource
- NEW: Informatie opvragen die nodig is om een resource aan te maken (formulieren, standaardwaarden)

Formaat URLs

REST URLs in WM3 zijn als volgt opgebouwd:

```
(url)/(prefix)/(resourcenaam)s/(resource_id)
```

Voorbeeld voor channel: `http://example.com/api/channels/1`

Is een resource afhankelijk van een andere resource, dan geldt dit formaat:

```
(url)/(prefix)/(resourcenaam1)s/(resource1_id)/(resourcenaam2)/new
```

Voorbeeld: Een record moet toegevoegd worden. Hiervoor moet het contenttype bekend zijn.

```
http://www.example.com/api/contenttypes/1/records/new
```

Voorbeeld voor Channel

Functie	Beschrijving	Link
GET	Een channel opvragen	http://www.example.com/api/channels/1
GET */new	Alle channels opvragen	http://www.example.com/api/channels
GET */new	Formulier opvragen	http://www.example.com/api/channels/new
POST	Channel aanmaken	http://www.example.com/api/channels
GET */edit	Formulieren opvragen voor een bestaand channel	http://www.example.com/api/channels/1/edit
PUT	Update van een channel	http://www.example.com/api/channels/1

Responses

WM3 retourneert altijd een JSON resultaat. Afhankelijk van de actie kan de structuur verschillen.
Succes: één resource (b.v. channel) opvragen

```
{
  "status": "success",
  "object": {
    "type": 42,
    "uri": "http://www.example.com",
    "language": "nl_NL",
    "id": 23
  }
}
```

Fout: een resource verwijderen

```
{
  "status": "error",
}
```

Bijlage REST API Reference

Channel

HTTP	Actie	Beschrijving
GET	channels/{id}	Channel opvragen
GET	channels	Alle channels opvragen

Layout

HTTP	Actie	Beschrijving
GET	layouts/{id}	Layout opvragen. Levert alle relevante informatie van een layout inclusief de slots en subslots (tree) en hun configuraties.

Widget

HTTP	Actie	Beschrijving
GET	widgets/{id}	Widget opvragen. Levert widget + configuratie + de widgettype
GET	widgettypes/{widggettype_id}/widgets/new	Levert het formulier voor het aanmaken van een widget op basis van een (benodigd) widgettype
POST	widgettypes/{widgettype_id}/widget	Een nieuw widget aanmaken op basis van een (benodigd) widgettype
GET	widgets/{id}/edit	Levert een formulier voor het updaten van een widget
PUT	widgets/{id}	Wijzigt een widget

Page

HTTP	Actie	Beschrijving
GET	pages/{id}	Levert een page
GET	pages/new	Levert een formulier voor het aanmaken van een nieuwe page
POST	pages	Maakt een nieuw page aan
GET	pages/{id}/edit	Levert het formulier voor het updaten van een page
PUT	pages/{id}	Wijzigt een page
DELETE	pages/{id}	Verwijdert een page

PageWidget (page<>widget relatie)

HTTP	Actie	Beschrijving
GET	pagewidgets/{id}	Levert informatie over een page<>widget relatie
GET	pagewidgets/new	Levert een formulier voor het aanmaken van een page<>widget relatie
POST	pagewidgets	Maakt een nieuwe page<>widget relatie aan
GET	pagewidgets/{id}/edit	Levert het formulier voor het updaten van een page<>widget relatie

PUT	pagewidgets/{id}	Wijzigt een page<>widget relatie
DELETE	pagewidgets/{id}	Verwijdert een page<>widget relatie

ContentBundle

ContentType

HTTP	Actie	Beschrijving
GET	contenttypes/{id}	Levert een contenttype
GET	contenttypes/new	Levert een formulier voor het aanmaken van een contenttype
POST	contenttypes	Maakt een nieuwe contenttype aan
GET	contenttypes/{id}/edit	Levert het formulier voor het updaten van een contenttype
PUT	contenttypes/{id}	Wijzigt een contenttype
DELETE	contenttypes/{id}	Verwijdert een contenttype

ContentField

HTTP	Actie	Beschrijving
GET	contentfields/{id}	Levert een contentfield
GET	contentfieldtypes/{id}/ contentfields/new	Levert het formulier voor het aanmaken van een nieuw contentfield op basis van een contentfieldtype
POST	contentfieldtypes/{id}/ contentfield	Maakt een nieuw contentfield aan op basis van een contentfieldttype
GET	contentfields/{id}/edit	Levert het formulier voor het updaten van een contentfield
PUT	contentfields/{id}	Wijzigt een contentfield
DELETE	contentfields/{id}	Verwijdert een contentfield

ContentFieldType

HTTP	Actie	Beschrijving
GET	contentfieldtypes/{id}	Levert een contentfieldttype
GET	contentfieldtypes	Levert alle contentfieldttypes

Record

HTTP	Actie	Beschrijving
GET	records/{id}	Levert een record
GET	contenttypes/{id}/records/new	Levert het formulier voor het aanmaken van een new record van een specifiek contenttype
POST	/contenttypes/{id}/record	Maakt een nieuw record van een specifiek contenttype
GET	records/{id}/edit	Levert het formulier voor het updaten van een specifiek record

PUT	records/{id}	Wijzigt een record
DELETE	records/{id}	Verwijdert een record

Bijlage ContentFieldTypes

ContentFieldTypes

Een Content field type in WM3 is een definitie van een nieuw data type. In de praktijk zijn content field types de velden die je in een standaard formulier ziet. Voorbeelden zijn tekstvelden, textareas of choices (radio/checkbox). Dit zijn basistypen die al in WM3 gedefineerd zijn.
Je kunt gemakkelijk nieuwe field types defineren. Deze field types mogen dan een concreet charakter hebben (zoals geboortedatuum, voornaam, achternaam, telefoon of email) of weer algemeen zijn (zoals text).
Content field types zijn nooit direct een content type gekoppeld of zijn al geconfigureerd (dit zouden dan content fields zijn). Je kunt ze daarom zien als prototypes over die je kunt beschikken als je een nieuw content type aanmaakt.

- ContentFieldType = Een prototype van een veld
- ContentField = Een veld/instantie voor een specifiek ContentType van type ContentFieldType
- ContentType = Een verzameling van ContentFields

Field types moeten altijd als service gedefineerd worden met een tag 'webstores.fieldtype' en van type FieldDefinitionInterface zijn (zie tags referentie):

```
 1
 2 <?php
 3
 4 namespace Webstores\Bundle\ContentBundle\Content\Definition\Type;
 5
 6 use Symfony\Component\Form\AbstractType;
 7
 8 /**
 9  * {@inheritdoc}
10  */
11 class FieldDefinition implements FieldDefinitionInterface
12 {
13     private $baseFormType;
14     private $configurationFormType;
15
16     public function __construct(AbstractType $baseFormType, AbstractType
       $configurationFormType)
17     {
18         $this->baseFormType = $baseFormType;
19         $this->configurationFormType = $configurationFormType;
20     }
21
22     /**
23      * @return Symfony\Component\Form\AbstractType
24      */
25     public function getBaseFormType()
26     {
27         return $this->baseFormType;
28     }
29
30     /**
31      * @return Symfony\Component\Form\AbstractType
32      */
33     public function getConfigurationFormType()
34     {
35         return $this->configurationFormType;
36     }
37 }
38
```

Hiervan een fieldtype 'Text' te defineren is simpel:

```
	<service
		id="webstores.contentbundle.field.text.configuration"
		class="%webstores.contentbundle.field.text.configuration_class%">

		<argument type="service"
			id="webstores.contentbundle.field.basic.configuration" />
	</service>
	<!-- ... -->
	<service
		id="webstores.contentbundle.field.text"
		class="Webstores\ContentBundle\Content\Definition\Type\FieldDefinition">

		<argument
			type="service"
			id="form.type.text"/>

		<argument type="service"
			id="webstores.contentbundle.field.text.configuration"/>

		<tag name="webstores.fieldtype"/>
	</service>
```

Beide argumenten (form.type.text en webstores.contentbundle.field.text.configuration) zijn services van type AbstractType.

Conclusie

Een fieldtype is op zich een gespan van twee abstract types: Ten eerste de type zelf. Ten tweede het configuratie formulier voor deze type. Een field type heeft altijd bepaalde eigenschappen (textvelden hebben max_length, een label enz). Wil je een contentfield aanmaken, moet je de type configureren. Daarom ben je altijd ook een bijhorend configuratie form type nodig!
Bijlage Conventies Services
Symfony2 biedt via de service container en DI een krachtige benadering om onderdelen van elkaar onafhankelijk te maken.
In plaats van direct een object aan te maken wordt gebruik gemaakt van een service container

```
class HelloController extends Controller
{
    // ...

    public function sendEmailAction()
    {
        // ...
        $mailer = $this->get('foo.bar.myfoo.mybar.ma1ler.bla');
    }
}
```

In het bovenstaande voorbeeld wordt een service met de naam foo.bar.myfoo.mybar.ma1ler.bla gebruikt. Hoewel dit flexibel is, zou je waarschijnlijk vaak de goede naam voor deze service opzoeken.
Daarom is een bepaalde conventie belangrijk. Door een logische conventie wordt dus het opzoeken van de benamingen vermeden. Voor WM3 gelden de volgende conventies:

- Eigen services hebben altijd een prefix webstores
- Services, die een interface binnen een bundle implementeren, hebben altijd de prefix: <company>.<bundle>. (voorbeeld: acme.hellobundle.blogentry.manager)
- Als derde parameter moet een context van de service opgegeven worden. Een context kan

een model zijn (derde parameter) (channel, page, widget, slot) of/en een bepaald omvangrijker onderdeel van de bundle zijn (mailer, renderer, parser) (derde/vierde parameter)

- De volgende parameter beschrijft de functionaliteit van dit bepaalde onderdeel. Dit moet duidelijk de functionaliteit van deze service beschrijven.
- Optioneel: Zijn er meerdere implementaties beschikbaar, dan moet er één implementatie de hoofdservice zijn (b.v.: webstores.corebundle.layout.loader). De andere implementaties moeten dezelfde naam hebben + een underscore + een specifiekere beschrijving. B.v.: webstores.corebundle.layout.loader_dummy

Bijlage: FormTypes en FormHandlers

Waarom FormHandlers en FormTypes?

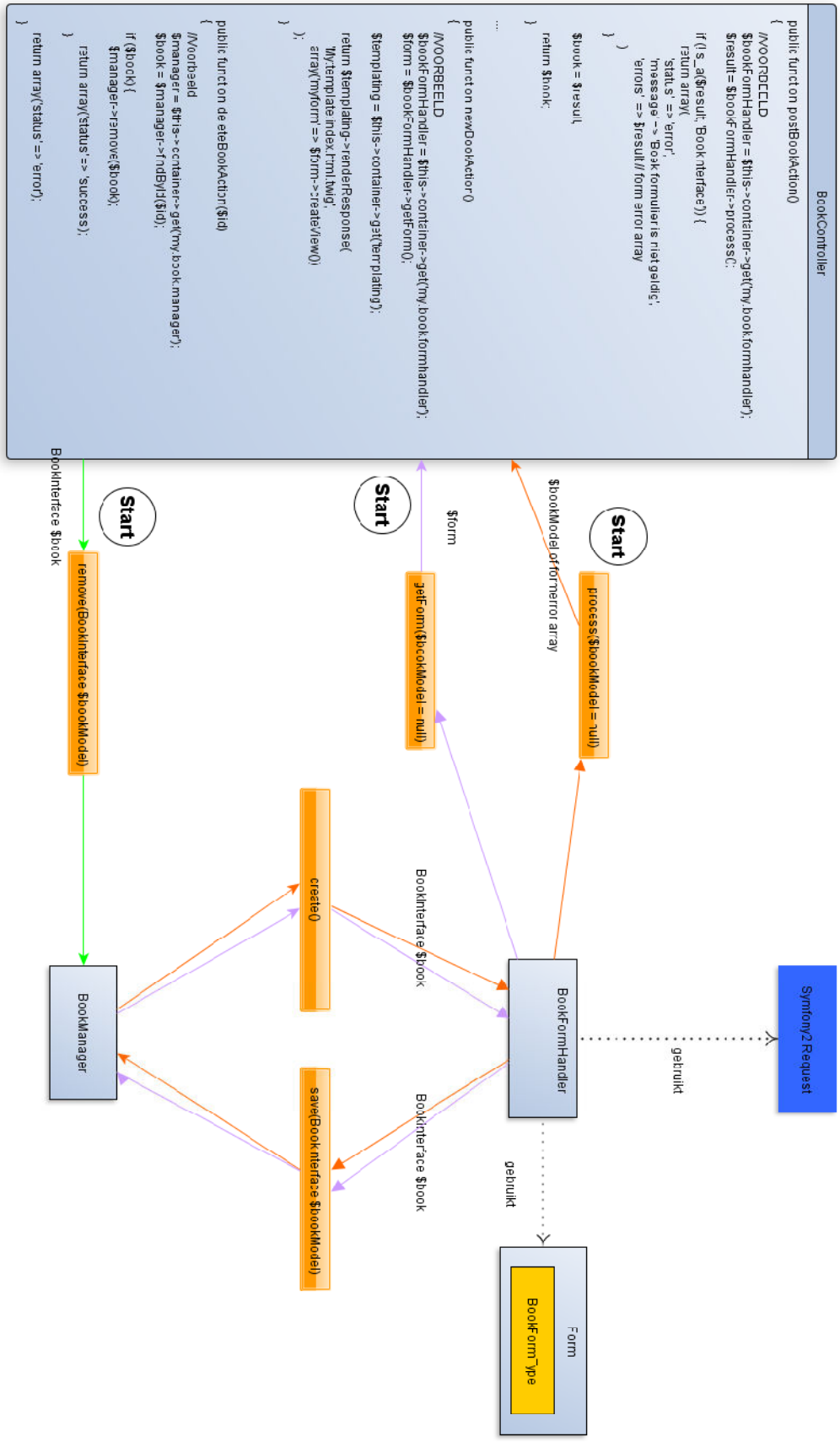
De taak van een FormHandler is het afhandelen van inkomende form requests. Met naam worden nieuwe models aangemaakt of gewijzigt door deze formulieren. De formhandler maakt een formulier aan een gebruikt hiervoor een FormType. FormHandler en FormType overnemen dus taken die allemal ook in de controller kunnen gebeuren. Toch heeft het outsourcen van formulieren voordelen:

- Stel er zijn merdere BookControllers (REST API BookController, FrontEnd BookController enz.). Stel, dat het boek model uitgebreid wordt met een nieuw veld datum_publicatie. Dit heeft als gevolg, dat de form afhandeling in alle BookControllers aangepast moeten worden - niet goed! Door deze benadering moet alleen de FormType aangepast worden

- Request objecten en manager objecten kunnen in formhandlers willekeurig met elkaar gecombineerd worden.

- Overzichtelijkheid van controllers en form handlers: Controllers en form handlers blijven heel schoon, omdat de taken verspreid worden

Volgende figuur is handig om de afhandeling van requests te begrijpen:



Bijlage: Hoe worden pagina's getoond?

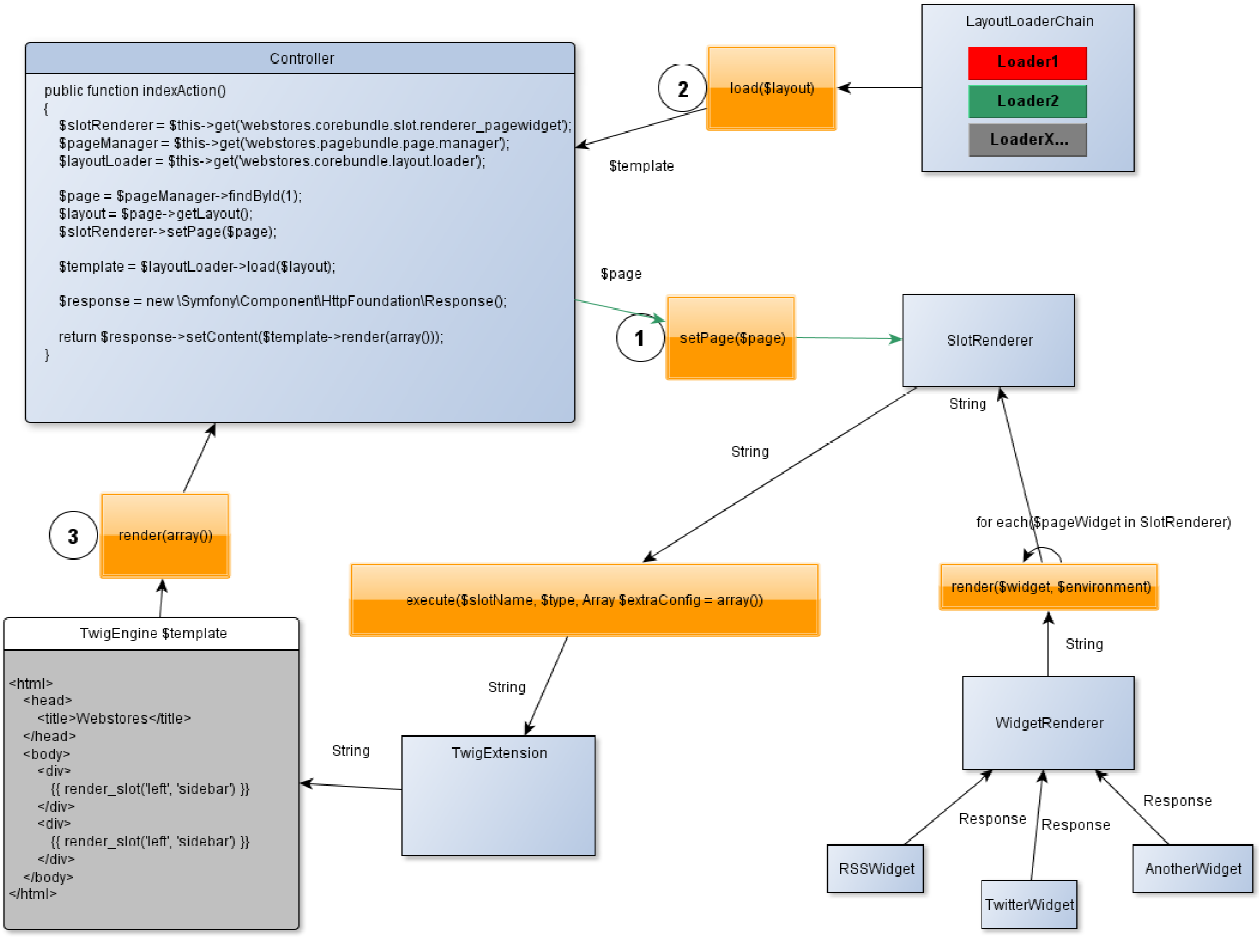
Het is handig om te weten, hoe WM3 pagina's toont. In onderstaande grafiek zie een flow van de code die uitgevoerd wordt, wanneer een pagina met een specifiek layout getoond wordt.

Benodigde informatie voor het tonen van een pagina:

- **Page**: De eerste stap is het opvragen van een pagina via de manager
- **Layout**: Aan ieder page is een layout gekoppeld (kan via $page->getLayout() opgevraagd worden)
- **PageWidgets**: Beschikbare widgets voor een specifiek page (kan via $page->getPageWidgets() opgevraagd worden)

Bovendien zijn er merdere classen nodig die het proces daadwerkelijk uitvoeren

- **Controller**: Kan als startpunt dienen
- Een **TwigExtensie**: Een twig extensie, die als bridge tussen het daadwerkelijke twig template en WidgetSlotRenderer dient.
- Een **SlotRenderer**: Een WidgetSlotRenderer heeft kennis van de actuele pagina de actule slot en kan over widgets beschikken. Dit object gebruikt een WidgetRenderer om een widget uiteindelijk uit te voeren.
- Een **WidgetRenderer**: Een WidgetRenderer weet, hoe uit een widget een string gemaakt wordt



Stap 0: Ophalen van benodige services
Om een pagina te tonen zijn minimaal een SlotRenderer, PageManager en een LayoutLoader nodig

Stap 1: Configuratie van slotRenderer
Voordat een twig template uitgevoerd wordt, moet ten eerste een page voor SlotRenderer gezet worden. Dit moet gebeuren voordat twig het resultaat rendert.

Stap 2: Ophalen van het layout
Via een layout model moet een fysiek layout opgehaald worden. Dit layout kan bij voorbeeld ergens op een hardeschrijf liggen of in een database staan.
Via een ChainLoader wordt geprobeerd dit fysieke layout te op te halen. De bron van een layout kan dus variieren. Geretourneerd wordt een Twig_Engine object, dat in de volgende stap gebruikt wordt om het eindresultaat te te tonen.

Stap 3: Renderen van het resultaat
Een simpel $template->render(array()) initiieert het rendering proces. Twig voert het template uit en komt op een gegeven moment twig-functies render_slot tegen. Daadwerkelijk voert Twig nu de WM3 twig extensie uit die wederom een of merdere SlotRenderers uitvoert. Een SlotRenderer kan van alles zijn en heeft als taak het retourneren van content voor een specifiek slot.
In dit voorbeeld wordt een SlotRenderer gebruikt die op de hoogte van het te tonen page object is en content van widgets dient te retourneren. De SlotRenderer zoekt alle widgets, die in het actuele slot getoond moeten worden en voert ze via een WidgetRenderer uit. De WidgetRenderer retourneert als laatste stap het widget resultaat als string.

Twig voert dus alle render_slot methoden uit die hij tegenkomt en retourneert string als resultaat. Deze string is de uiteindelijke (html) pagina met een passend layout en alle bijhorende widgets.
Eigen SlotRenderers

Er is een mogelijkheid heel simpel eigen slotrenderers aan te maken. Het CoreBundle (Channels, layouts en slots) kan onafhankelijk van het PageBundle en WidgetBundle gebruikt worden. In feite maakt met naam het PageBundle gebruik van het CoreBundle en implementeert een eigen SlotRenderer om widgets in slots te tonen.
Je kunt dus het CoreBundle stand-alone gebruiken.
Om een eigen slotrenderer aan te maken, hoef je alleen een service van de het interface SlotRendererInterface met de Tags webstores.slotrenderer registreren. Meer hoef je niet te doen om eigen inhoud in slots te plaatsen.

Bijlage: Records

Als je het datamodel van het ContentBundle bekijkt, zie je geen Record tabel. Dit komt, omdat Records in een "virtuele", dynamische tabel verticaal worden opgeslagen. Stel, er was een virtuele Gebruikers tabel:

Records opvragen en bewerken doe je op de standaard manier met een manager. De benodigde id is de id van de contenttabel:

```
 //...
$jamesBondRecord = $recordManager->findById(42);
echo 'My name is ' . $jamesBondRecord['naam'].', '.$jamesBondRecord['voornaam'].' '.
        $jamesBondRecord['naam'].'.';

$jamesBondRecord['naam'] = 'Bond Junior';
$recordManager->save($jamesBondRecord);
```