in2code.
Wir leben TYPO3

Entwickeln und warten von TYPO3-Erweiterungen

in TYPO3 9.5

Einführung

Grundlegendes

Nachfolgende Dokumentation inklusive Beispiele und Screenshots beziehen sich ausschließlich auf **TYPO3 9LTS**. Da die Entwicklung von TYPO3 in großen Schritten vorangetrieben wird, ergeben sich immer wieder Änderungen im Laufe der Zeit. Die Rückwärtskompatibilität ist daher nicht immer gewährleistet (Breaking Changes).

So ist z.B. der Einsatz von $GLOBALS['TYPO3_DB'] mit den kompletten Datenbankfunktionen ab TYPO3 9 nicht mehr möglich. Eine Alternative mit Unterstützung von Doctrine bietet beispielsweise der QueryBuilder.

Das bedeutet auch, dass eine Erweiterung, die unter TYPO3 7.6 erstellt wurde, sehr wahrscheinlich nicht oder nur teilweise unter TYPO3 9.5 lauffähig ist und umgekehrt.

TYPO3 besteht zum größten Teil aus PHP, HTML, CSS und JavaScript. Ergänzt wird dies durch Konfiguration in TypoScript oder Yaml. Beim Entwickeln neuer Plugins wird ein grundlegendes Verständnis dieser Sprachen vorausgesetzt.

Die Dokumentation ist vor allem für die Nutzung von Linux oder Mac erstellt worden. Unter Windows gibt es mögliche Abweichungen.

TYPO3 9.5 LTS

Begrifflichkeiten

Extbase

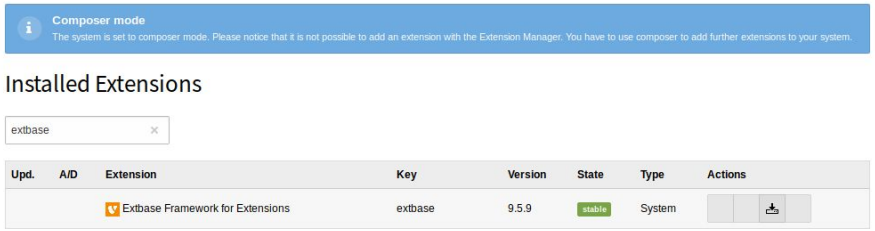


Abb: Extension Manager mit Filterung nach “extbase”

Tipp: Die Extension Extbase ist in TYPO3 bereits standardmäßig installiert und aktiviert

Bei **Extbase** handelt es sich um eine Erweiterung in TYPO3. Mit Hilfe dieses “kleinen PHP-Frameworks” ist es möglich, Erweiterungen zum TYPO3 in einer modernen Programmierweise zu schreiben. Es handelt sich also um eine Schicht zwischen TYPO3 und der zu entwickelten Extension. Extbase ermöglicht komplexe Vorgänge und das mit einem Minimum an zu schreibenden Code.

Entstanden ist Extbase ursprünglich aus einem kleinen Backport aus dem FlowFramework (ursprünglicher Name TYPO3 Flow). Im Vergleich mit anderen PHP-Frameworks (wie z.B. Symfony oder Laravel) kann Extbase natürlich nicht mithalten, es lassen sich jedoch deutliche Parallelen erkennen.

Fluid

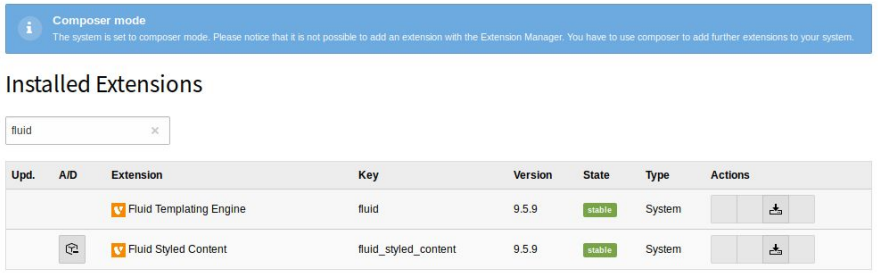


Abb: Extension Manager mit Filterung nach “fluid”

Tipp: Fluid ist nicht zu verwechseln mit Fluid Styled Content. Letzter kümmert sich um das Markup von Seiteninhalten in TYPO3 mit Hilfe der Extension Fluid

Bei **Fluid** handelt es sich um eine eigene, mächtige Template-Engine, die als Package vom TYPO3 CMS und anderen Frameworks (FlowFramework, Neos CMS) genutzt werden kann.

Als Brücke zwischen TYPO3 und diesem Paket befindet sich die TYPO3-Erweiterung fluid, die das Standalone-Package (siehe Verzeichnis vendor/typo3fluid) einbindet und um weitere Funktionen (wie z.B. den ViewHelper f:link.typoolink) ergänzt.

Die alte Template Engine von TYPO3 (cObject TEMPLATE) gilt als überholt und eignet sich nicht im Einsatz mit MVC, da diese nicht mit Objekten, Schleifen, Bedingungen umgehen kann.

Fluid wird mittlerweile nicht nur für das Rendering von Extensinos benutzt, sondern kann auch direkt in der TYPO3-Integration verwendet werden. Dies geht mit dem cObject **FLUIDTEMPLATE** (siehe TSref oder Administrationsschulungsunterlagen von in2code).

Abstract Plugin / piBase Extension

Unter dem umgangssprachlichen Begriff einer **piBase-Extension** versteht man TYPO3-Erweiterung, die bei der Einbindung nicht auf die Methoden und Klassen von Extbase zurückgreift. Der offizielle Begriff hier ist übrigens Abstract Plugin.

Bei einer Vielzahl der verfügbaren, freien Erweiterungen im TYPO3 Extension Repository (TER) handelt es sich noch um solche, oftmals in die Jahre gekommenen Extensions.

Objektorientierte Programmierung

Mit Extbase hält die **objektorientierte Programmierung** Einzug in TYPO3. Bei **OOP** handelt es sich um ein Programmierparadigma mit der Grundidee, Daten und Funktionen möglichst eng in einem sogenannten Objekt zusammenzufassen, so dass Methoden fremder Objekte diese Daten nicht versehentlich manipulieren können.

OOP steht im Gegenzug zur **Prozeduralen Programmierung**. Hier wird ein Programm von Anfang bis Ende zeilenweise durchlaufen.

Als Grundlage gilt es die Wirklichkeit möglichst einfach, verständlich und originalgetreu in Objekten nachzubauen. Hierbei gilt:

- Alles wird als Objekt gesehen
- Objekte stehen in Verbindung zueinander
- Jedes Objekt hat Attribute und Methoden
- Objekte können voneinander erben
- Jedes Objekt kann bestimmen, wie von außen auf die Attribute und Methoden zugegriffen werden kann

Klassen sind dabei der Bauplan für ein Objekt. Die Klasse definiert, welche Eigenschaften und Möglichkeiten ein Objekt hat. Das Objekt ist die individuelle Ausprägung dieser Klasse.

OOP: Aus der realen Welt - das Auto-Beispiel

Abb: Fahrzeug

Sie erhalten die Aufgabe, eine Software zu entwickeln, mit der man ein Fahrzeug steuern können soll. Auch wenn PHP nicht unbedingt die perfekte Wahl ist, wäre es dennoch möglich.

Im ersten Schritt liegt der Fokus in der Beschleunigungs- und Scheinwerfersteuerung.

Eigenschaften (Attribute)	Funktionen (Methoden)
Scheinwerferstatus (an/aus) “lights” (boolean)	Scheinwerfer aktivieren oder deaktivieren auf Knopfdruck
Geschwindigkeit (in km/h) “speed” (integer)	Beschleunigen oder Abbremsen bzw. zu voreingestellter Geschwindigkeit wechseln (Tempomat)

```
class car
{
    protected $lights = false;

    protected $speed = 0;

    public function accelerate(): void
    {
        $this->speed += 5;
    }

    public function decelerate(): void
    {
        $this->speed -= 10;
        if ($this->speed < 0) {
            $this->speed = 0;
        }
    }

    public function setSpeedControl(int $speed = 50): void
    {
        $this->speed = $speed;
    }

    public function getSpeed(): int
    {
        return $this->speed;
    }

    public function switchLights(): void
    {
        if ($this->lights === false) {
            $this->lights = true;
        } else {
            $this->lights = false;
        }
    }
}
```

Die Klasse **car** hält zwei für uns wichtige Eigenschaften (**lights** mit der Grundeinstellung „false“ und **speed** mit “0”). Diese beiden Eigenschaften sollen aus Sicherheitsgründen nicht von außen veränderbar sein, daher wurden diese als protected gekennzeichnet. Über Methoden in unserer Klasse lassen sich die Eigenschaften definiert manipulieren.

Über **accelerate**() erhöht sich die aktuelle Geschwindigkeit um den Wert 5. Mit **decelerate**() reduziert sie sich um 10 (hier ist bereits eine sicherheitsfunktion eingebaut, damit das Auto nicht plötzlich rückwärts fährt). Bei **setSpeedControl**() wird die Geschwindigkeit sofort auf 50 gestellt, falls nicht anders übergeben. Mit dem Getter **getSpeed**() erhalten wir die aktuelle Geschwindigkeit (z.B. für eine Ausgabe im Tachometer).

Daneben lässt sich das Licht durch einen Knopfdruck an- oder ausschalten. Hierzu muss lediglich die Methode **switchLights**() aufgerufen werden. Aus Platzgründen haben wir in dem Beispiel auf eine zweite Getter-Funktion **getLightStatus**() verzichtet. Diese wäre aber notwendig, um die Lichter anzuschalten.

Alle diese Methoden sind als public ausgezeichnet und daher von außen aufrufbar.

Zugriffsregelungen - Deklarationen von Attributen und Methoden:

public	Direkter Zugriff von außen möglich
protected	Interner Zugriff, Zugriff von Tochterklassen aus möglich
private	Nur interner Zugriff

In der OOP werden Attribute in einem Modell nicht als public deklariert. Eine Änderung und das Auslesen erfolgen ausschließlich über Getter- und Setter-Methoden. Nur so können übergreifende Logiken eingebaut und erhalten werden.

Model View Controller (MVC)

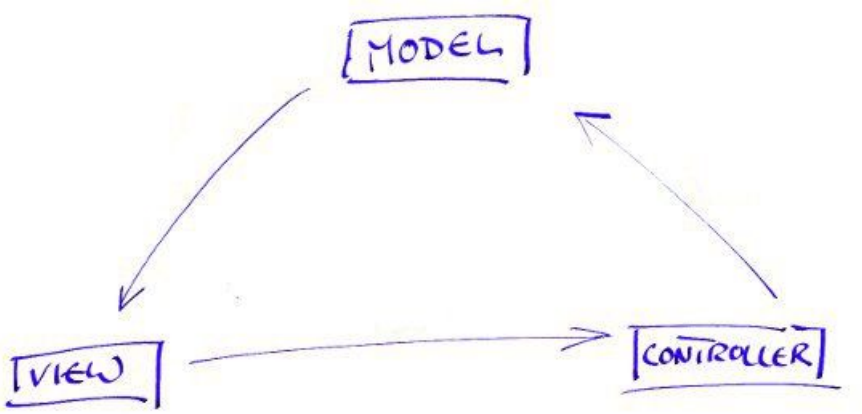


Abb: MVC-Konzept

Model View Controller (MVC) ist ein Muster zur Strukturierung von Software-Entwicklung in drei Einheiten. Ziel ist hierbei ist ein flexibler Programmentwurf, der eine spätere Änderung oder Erweiterung möglichst erleichtert. Die Wiederverwendbarkeit der Programmierung steht also deutlich im Vordergrund. Extbase bringt ebenfalls das MVC-Konzept in die TYPO3-Extension–Entwicklung.

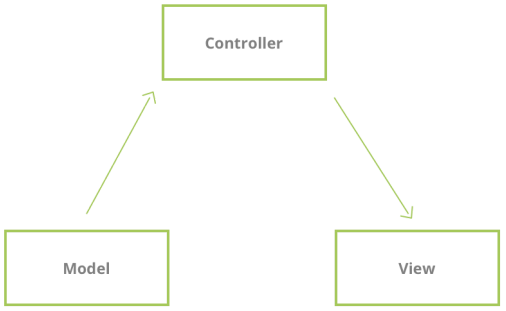


Abb: MVC mit einem View

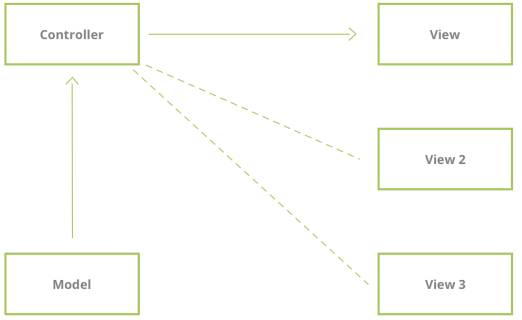


Abb: MVC mit mehreren Views

In dem oben abgebildeten Skalierungsbeispiel sieht man die Erweiterung des Views um weitere Ausgaben. Man könnte beispielsweise die HTML-Listenansicht ganz einfach um einen RSS-Feed und eine JSON-Ausgabe erweitern, ohne an der Logik der Extension Änderungen vorzunehmen zu müssen.

Convention over Configuration

Der wohl verständlichste Vorteil von Extbase im Vergleich zum Abstract Plugin ist die Reduzierung der Code-Komplexität durch vorgegebene Konventionen (**Konvention vor Konfiguration**).

Ein einfaches Beispiel hierzu ist der physikalische Ort eines HTML-Templates. Während es früher dem Entwickler völlig frei gestellt war, wo und ob er eine HTML-Datei mitliefert, erwartet eine Action in einem Extbase-Controller bereits eine Datei mit einer bestimmten Bezeichnung an einem bestimmten Ort. Der Entwickler muss sich also nicht auch noch um die Einbindung und Verarbeitung der Template-Dateien kümmern.

Selbstverständlich lässt sich dieses Verhalten auch aufbrechen und weiter individualisieren, ergibt aber nur in seltenen Fällen wirklich Sinn.

Tipp: Ganz nebenbei findet man sich auch in fremden Erweiterungen viel schneller zurecht als früher, da man bereits weiß, wo sich welche Dateien befinden sollten.

Domain Driven Design

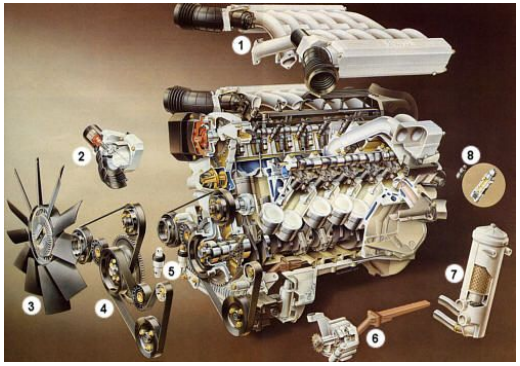


Abb: Abstraktion eines Bauplans

Domain-Driven Design (DDD) ist nicht nur eine Technik oder Methode. Es ist vielmehr eine Denkweise und Priorisierung zur Steigerung der Produktivität von Softwareprojekten im Umfeld komplexer fachlicher Zusammenhänge.

Der Sinn jeder Software ist es, die Aufgabenstellungen einer bestimmten Anwendungsdomäne zu unterstützen. Um dies erfolgreich leisten zu können, muss die Software harmonisch zu der Fachlichkeit der Anwendungsdomäne passen, für die sie bestimmt ist. Domain-driven Design ermöglicht dies, indem die Software grundlegende Konzepte und Elemente der Anwendungsdomäne sowie deren Beziehungen modelliert.

Die Abstraktion der Wirklichkeit in realitätsnahen Objekten wird ebenfalls mit Extbase unterstützt. So wäre es theoretisch denkbar gemeinsam mit dem Kunden das Model aufzusetzen und hierbei wieder auf Begrifflichkeiten aus der wirklichen Welt zurückzugreifen.

Vendor

Mit der Einführung von Namespaces (siehe nachfolgenden Bereich) wurden auch Vendor-Namen eingeführt. Der Vendor Name ist der kleinste gemeinsame Nenner zwischen Extensions aus einer Hand. Üblicherweise nutzen einzelne Entwickler oder Gruppierungen einen fixen Vendor (z.B. „In2code“ für Packages/Extensions der in2code GmbH).

Somit kann das Package mit dem Namen „News“ von mehreren Anbietern entwickelt zu werden – z.B. UmbrellaCorporation/News

Tipp: Auf packagist.org kann man nach frei verfügbaren PHP-Paketen für die Installation mit composer suchen. Dort besteht der Name eines Paketes immer aus “Vendor/Paketname”

Namespaces in PHP

Namespaces - also der **Namensraum** - gibt es im TYPO3-Kontext an mehreren Stellen. Neben der Deklaration von ViewHelpern in Fluid und Bereichen für die Einbindung von JavaScript-Dateien über require.js interessieren wir uns vor allem um die **Bereichsabtrennung im PHP**.

Würde man eine Klasse lediglich “car” nennen, ist die Wahrscheinlichkeit groß, dass es noch eine zweite gleich benannte Datei gibt. PHP wüsste hierbei nicht, welche Klasse geladen werden soll. Durch einen Präfix mit Vendor und Paketname oder Extensionname wird das Problem umgangen.

Per Konvention in Extbase ergibt sich aus dem Pfad und Dateiname der komplette Klassenname (full qualified name) und umgekehrt: (Vendor\ExtensionName\Verzeichnis\Datei => typo3conf/ext/extkey/Classes/Verzeichnis/Datei.php)

Klassenname	Pfad/Datei
\Vendor\Extension \Domain\Model\Car	EXT:extension/Classes/Domain/ Model/Car.php

Tipp: Bei Dateien ohne Namespace-Deklaration handelt es sich meist noch um Relikte aus der Zeit vor TYPO3 6.0

Tipp: Achten Sie auf korrekte Schreibweisen. Andernfalls können Dateien nicht per Autoloader geladen werden!

Composer

Zur Installation von TYPO3 empfiehlt sich auf jeden Fall der Einsatz des PHP-Package-Managers Composer. Über eine JSON-Konfiguration (composer.json) wird TYPO3 mit allen Abhängigkeiten per Befehl auf der Konsole heruntergeladen und zusammengebaut (Build-Prozess).

Einmal gebaut, wird das Ergebnis in einer composer.lock Datei festgehalten (dieses Verhalten lässt dann auch gut in einem Deployment-Szenario verwenden).

Hinweis: Wird TYPO3 nicht über Composer installiert, läuft TYPO3 im "**Classic Mode**" andernfalls im "**Composer Mode**". Dies lässt sich auch beim Aufruf des Extension Managers ablesen.

```
{
  "repositories": [
        {
        "type": "composer",
        "url": "https://composer.typo3.org/"
        }
  ],
  "require": {
        "typo3/minimal": "9.5.*",
        "georgringer/news": "7.2.*"
  }
}
```

Tipp: Über composer lässt sich die TYPO3 ganz individuell zusammenbauen. Jede Systemextension ist ("Subtree-Split") in einem eigenen Paket untergebracht, welches man bei Bedarf hinzufügen kann. So bleibt das TYPO3 schlank, schnell und sicher

PSR Normen in PHP

Eine PHP Standard Recommendation (**PSR**) ist eine PHP-Spezifikation, welche durch die PHP Framework Interop Group veröffentlicht wird. Ziel ist es die Interoperabilität von Komponenten zu ermöglichen und eine gemeinsame technische Basis zu schaffen. Verschiedene Frameworks wie z. B. die der **TYPO3-Community, Symfony oder Zend** implementieren hierbei PSR-Spezifikationen in einem selbst gewählten Umfang.

Diese Standards **werden bereits** von TYPO3 umgesetzt:

PSR-1, PSR-2	Coding Standards
PSR-3	Logging Interface
PSR-4 (PSR-0)	Autoloading Standard
PSR-7	HTTP message interfaces
PSR-14	Event Dispatcher
PSR-15	Middleware
PSR-17	HTTP Factories
PSR-18	HTTP Client

Siehe auch https://www.php-fig.org/psr/#numerical-index für eine Übersicht der Standards.

PSR Normen: PSR-1, PSR-2

Diese **PSR Spezifikationen** beinhalten Vorgaben zur Formatierung einer PHP-Datei.

Als wichtigste Merkmale in PSR-1/PSR-2 gelten: Einrückung mit 4 Leerzeichen, Öffnende Klammern in Funktionen und Klassen auf einer neuen Zeile, Klassenname ist identisch zum Dateinamen (ohne Endung) und keine schließenden PHP-Tags.

Tipp: Eine gute IDE (z.B. PhpStorm) hilft bei den Vorgaben. Über einen PHP-Codesniffer lässt sich dies auch noch automatisch abtesten (beim Entwickeln oder im Deployment).

```
<?php
declare(strict_types=1);
namespace Vendor\Extension\Domain\Service;

/**
 * Class DoSomethingService for magical things
 */
class DoSomethingService
{
    /**
     * @param string $string
     * @return string
     */
    public function trim(string $string): string
    {
        return trim($string);
    }
}
```

Schreibweise (CamelCase, Lower Underscored)

Innerhalb von TYPO3 lassen sich **verschiedene Schreibweisen** von Dateien, Methoden, Klassen oder Konfiguration etc... beobachten. Seit Extbase (und spätestens mit PSR-2) kommt CamelCase auf den Plan – also eine Schreibweise, bei der mehrere, zusammengesetzte Wörter, innerhalb eines Begriffs durch einen Großbuchstaben optisch voneinander abgegrenzt werden.

Historisch bedingt gibt es leider eine Vielzahl von Möglichkeiten, die teilweise auch etwas verwirrend sind. So z.B. in TypoScript und Datenbank drei verschiedene Schreibweisen (z.B. „ATagParams“, „bodyTagCObject“ oder „index_enable“ / bzw. in tt_content „CType“, „colPos“ und „fe_group“) um Rückwärtskompatibilität einzuhalten.

Schreibweisen	„Umbrella Corporation“	Vorkommen
Lower Underscored	umbrella_corporation	Tabellennamen, alte Methodennamen (z.B. Hooks)
Upper CamelCase	UmbrellaCorporation	Datei-, Verzeichnisnamen, Klassennamen.
Lower CamelCase	umbrellaCorporation	PHP Variablen- oder Methodennamen

Tipp: Achten Sie auf korrekte Schreibweisen. Andernfalls können Dateien nicht per Autoloader geladen werden!

Deployment

Unter Deployment versteht man in der Softwareentwicklung die **Verteilung von Software** auf verschiedene Zielsysteme. In der Webentwicklung möchte man üblicherweise lokal entwickeln, danach auf einem Testserver selber kontrollieren, auf einem Releasesystem vom Product Owner eine Abnahme erhalten und letztlich auf ein Produktivsystem final ausrollen, bevor der Prozess von vorne beginnt.

Das Ganze soll natürlich möglichst ohne Ausfall eines Systems passieren.

Verschiedene Software hilft einem beim Aufbau eines Continous Deployments (Task Runner wie TYPO3 Surf oder Deployer oder CI-Software wie Gitlab CI oder Jenkins).

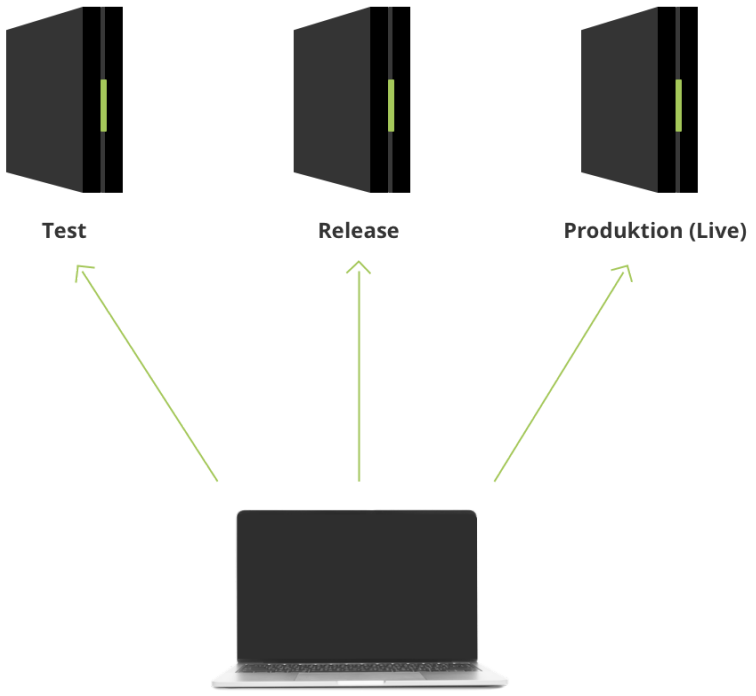


Abb: Abstraktion eines Deployment-Prozesses

Vorbereitung der Entwicklungsumgebung

Wir empfehlen eine lokale Entwicklung. Dies ermöglicht nicht nur eine Entwicklung ohne Internetverbindung sondern sichert auch, dass sich Entwickler nicht gegenseitig behindern.

Damit TYPO3 lokal läuft, benötigt es einige Vorbereitungen. Neben PHP muss auch Apache (oder eine Alternative wie Nginx) und MySQL (bzw. MariaDB) installiert und lauffähig sein.

Diese Voraussetzungen sind unter Linux beispielsweise nativ möglich. Unter Windows ist Xampp und unter Max Mamp ein beliebtes Tool. Eine noch bessere Alternative bietet der Einsatz von DDEV. Dieses kostenlose und auf Docker basierte Tool kümmert sich um einen echten Linux-Server als Unterbau.

DDEV

TYPO3 bietet selbst eine gute Dokumentation zum Aufsetzen - siehe https://docs.typo3.org/m/typo3/guide-contributionworkflow/master/en-us/Appendix/SettingUpTypo3Ddev.html

Zuerst muss Docker und DDEV installiert werden - siehe https://ddev.readthedocs.io/en/latest/

Anschließend kann man mit der eigentlich Installation loslegen:

```
mkdir newproject && cd newproject
composer require typo3/minimal
# optional: ergänzen .ddev/config.yaml “php_version: "7.3"”
ddev start
```

Oder ohne installiertem PHP und Composer:

```
mkdir newproject && cd newproject
git clone git://git.typo3.org/Packages/TYPO3.CMS.git .
# optional: ergänzen .ddev/config.yaml “php_version: "7.3"”
ddev start
```

Erste Ausgabe im Frontend

Nachdem TYPO3 installiert ist und man sich ins Backend eingeloggt hat, fehlt noch eine minimale Grundkonfiguration, damit man eine Ausgabe im Frontend erhält.

```
page = PAGE
page.10 < styles.content.get


page.20 = HMENU
page.20 {
    wrap = <hr />|
    1 = TMENU
    1.expAll = 1
    1.wrap = <ul>|</ul>
    1.NO = 1
    1.NO.wrapItemAndSub = <li>|</li>

    2 < .1
    3 < .1
}
```

Tipp: Nicht vergessen, im TypoScript das Static Template von fluid_styled_content dazu zu laden, damit normale Seiteninhalte ausgegeben werden können.

Best Practice bei der Entwicklung

Normalerweise möchte man, dass die lokale Entwicklungsumgebung anders eingestellt ist, als die eigentliche Liveumgebung. So möchte man lokal normalerweise ausführliche Fehlermeldungen, das Cachinge beeinflussen oder einfach auch bei längerer Inaktivität im Backend nicht gleich ausgeloggt sein.

Best Practice: AdditionalConfiguration.php

Es empfiehlt sich ein Verzeichnis **typo3conf/ext/AdditionalConfiguration/** anzulegen. Danach dieser Eintrag in der typo3conf/ext/AdditionalConfiguration.php

```
<?php
foreach (glob(__DIR__ .
'/AdditionalConfiguration/*Configuration.php') as $file) {
        include($file);
}
```

Best Practice: DevelopmentConfiguration.php

Über eine **typo3conf/ext/AdditionalConfiguration/DevelopmentConfiguration.php** können jetzt beispielsweise Caches deaktiviert und Fehlermeldungen aktiviert werden:

```
<?php
// PASSWORDS
if
(class_exists(\TYPO3\CMS\Core\Crypto\PasswordHashing\PasswordHas
hFactory::class)) {
	$hashInstance =
\TYPO3\CMS\Core\Utility\GeneralUtility::makeInstance(

\TYPO3\CMS\Core\Crypto\PasswordHashing\PasswordHashFactory::clas
s
	)->getDefaultHashInstance('BE');
	$GLOBALS['TYPO3_CONF_VARS']['BE']['installToolPassword']
= $hashInstance->getHashedPassword('pw');
} else {
	$saltFactory =
\TYPO3\CMS\Saltedpasswords\Salt\SaltFactory::getSaltingInstance(
);
	$GLOBALS['TYPO3_CONF_VARS']['BE']['installToolPassword']
= $saltFactory->getHashedPassword('pw');
}

// MISC
$GLOBALS['TYPO3_CONF_VARS']['SYS']['trustedHostsPattern'] = '*';
unset($GLOBALS['TYPO3_CONF_VARS']['BE']['cookieDomain']);
unset($GLOBALS['TYPO3_CONF_VARS']['SYS']['cookieDomain']);

$GLOBALS['TYPO3_CONF_VARS']['BE']['sessionTimeout'] =
9999999999;
$GLOBALS['TYPO3_CONF_VARS']['BE']['loginSecurityLevel'] =
'normal';
$GLOBALS['TYPO3_CONF_VARS']['BE']['fileCreateMask'] = '0775';
$GLOBALS['TYPO3_CONF_VARS']['BE']['folderCreateMask'] = '2775';
$GLOBALS['TYPO3_CONF_VARS']['BE']['lockSSL'] = '0';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['enableDeprecationLog'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlUse'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['curlTimeout'] = 10;

// DEBUG
$GLOBALS['TYPO3_CONF_VARS']['BE']['debug'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['BE']['compressionLevel'] = 0;
$GLOBALS['TYPO3_CONF_VARS']['FE']['debug'] = 1;
$GLOBALS['TYPO3_CONF_VARS']['FE']['compressionLevel'] = 0;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['displayErrors'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['devIPmask'] = '*';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['sqlDebug'] = '1';
\TYPO3\CMS\Core\Utility\ExtensionManagementUtility::addTypoScrip
t(
	'developmentConfiguration',
	'setup',
	'config.contentObjectExceptionHandler = 0'
);

// CACHE
$GLOBALS['TYPO3_CONF_VARS']['SYS']['clearCacheSystem'] = '1';
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['cache_hash']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['cache_pages']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['cache_pagesection']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['cache_phpcode']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['cache_rootline']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['extbase_datamapfactory_datamap']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['extbase_object']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['extbase_reflection']['backend']
	= \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
```

```
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['extbase_typo3dbbackend_queries']['backend']
       = \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['extbase_typo3dbbackend_tablecolumns']['backend']
       = \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['l10n']['backend']
       = \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
if
(!empty($GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConf
igurations']['news']['backend'])) {

$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['news']['backend']
       = \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
}
if
(!empty($GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConf
igurations']['news']['frontend'])) {

$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfiguratio
ns']['news']['frontend']
       = \TYPO3\CMS\Core\Cache\Backend\NullBackend::class;
}
$GLOBALS['TYPO3_CONF_VARS']['EXT']['extCache'] = '0';


// MAIL
$GLOBALS['TYPO3_CONF_VARS']['EXT']['powermailDevelopContextEmail
'] = 'testreceiver@mail.org';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromAddress'] =
'sender@mail.org';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['defaultMailFromName'] =
'defaultmailfromaddress';

// MAIL mbox
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport'] = 'mbox';
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_mbox_file'] =
'/var/www/html/mbox.txt';

// Mail smtp
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_server'] =
'sslout.de:465';
```

```
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_encrypt']
= 'ssl';
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_username']
= 'test@mail.org';
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_password']
= 'abcdef123456';
//$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_port'] =
'465';

// MAIL mailhog
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport'] = 'smtp';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_server'] =
'127.0.0.1:1025';
$GLOBALS['TYPO3_CONF_VARS']['MAIL']['transport_smtp_port'] =
'1025';
```

Tipp: Oftmals unterscheidet sich die DevelopmentConfiguration.php nicht zwischen den Instanzen und kann daher auch zentral abgelegt und versymlinkt werden.

Einrichtung der IDE (PhpStorm)

Beim Entwickeln mit TYPO3 werden in erster Linie PHP-, HTML-, JavaScript- und XML-Dateien erzeugt und bearbeitet. Dies lässt sich auch mit einem einfachen Editor bewerkstelligen.

Entscheidet man sich jedoch für eine IDE (Integrated Development Environment), stehen einem eine große Auswahl an Hilfsfunktionen zur Verfügung, die die Dauer der Entwicklung signifikant senken und die Codequalität erhöhen können.

Bekannte IDE-Software sind Netbeans, Eclipse oder PhpStorm. Eine Vielzahl von TYPO3-Core- und Extension-Entwickler haben sich bereits für PhpStorm entschieden. Nachfolgende Beispiele setzen daher auf PhpStorm (http://www.jetbrains.com/phpstorm/) auf.

Tipp: Die Lizenzkosten dieser Anwendung sind überschaubar. Wer jedoch bereits an einem Open-Source Projekt arbeitet, hat die Möglichkeit, eine freie Lizenz zu ergattern. Wer die Software erst einmal testen möchten, kann den kostenlosen 30-Tage Zeitraum nutzen.

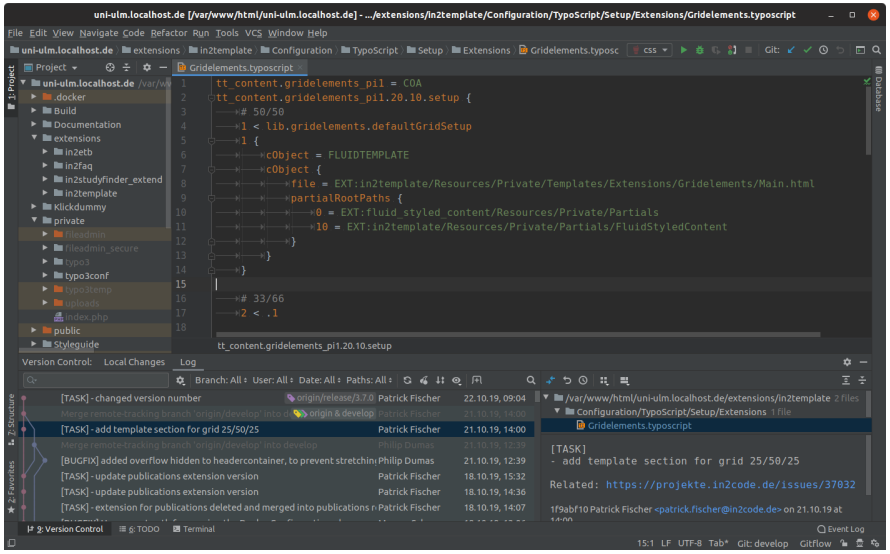


Abb: Beispielansicht eines TYPO3-Projekts in der IDE PhpStorm

Live Templates zum einfacheren Debuggen

TYPO3 unterstützt verschiedene Debugging-Methoden. Diese lassen sich durch ein Tastenkürzel in PhpStorm einbinden. Um dies einzurichten, muss man File/Settings/Live Templates und dann PHP bzw. HTML auswählen.

Empfohlene Live Templates:

Kürzel	Template Text	Dateityp
vd	\TYPO3\CMS\Extbase\Utility\DebuggerUtility::var_dump(var, 'vendor: ' . __CLASS__ . ':' . __LINE__);	PHP
vd	<f:debug>var</f:debug>	HTML

Tipp: Natürlich unterstützt PhpStorm auch professionelles Debugging mit z.B. xdebug. Hierzu muss der Server entsprechend vorbereitet werden

PhpStorm Plugin TypoScript

Wird viel TypoScript bei der Extension Entwicklung oder bei der Integration verwendet, so ist das „TypoScript - Enterprise“ von sgalinski Internet Services sehr nützlich.

Nach Einbindung gibt es Autovervollständigung und Codeanalyse für TypoScript in entsprechenden Dateien.

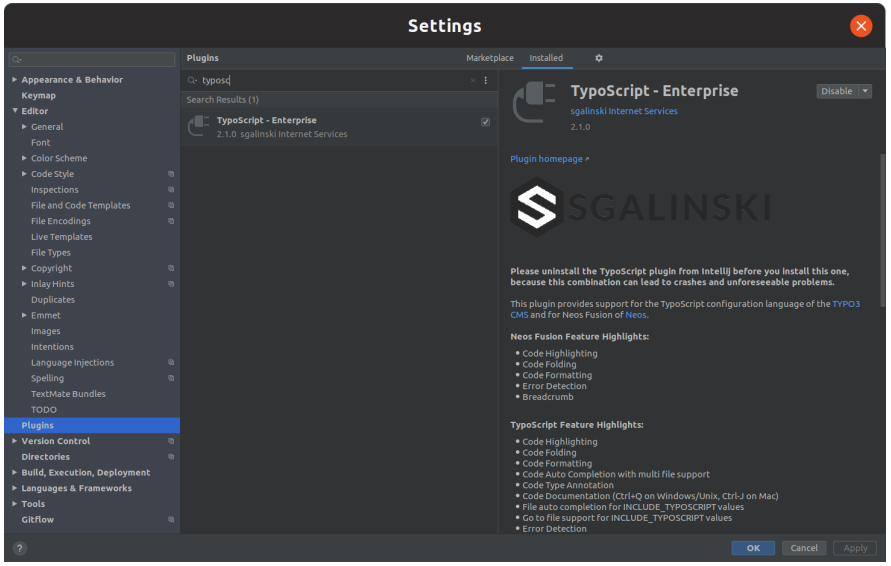


Abb: Installation des PhpStorm Plugins TypoScript

PhpStorm Plugin Fluid

Um Autovervollständigung in FLUID-Templates genießen zu können, empfiehlt sich das Plugin „Fluid - Enterprise“.

Nach einer Installation ist eventuell ein Neustart von PhpStorm notwendig.

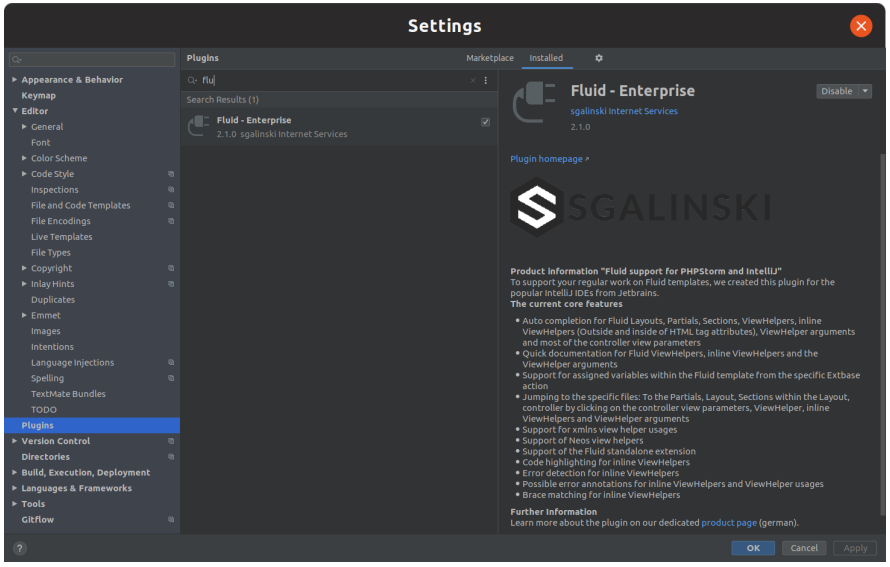


Abb: Installation des PhpStorm Plugins Fluid

PhpStorm Plugin TYPO3 CMS

Damit die IDE erkennt, dass es sich beim Rückgabewert von GeneralUtility::makeInstance() oder ObjectManager->get() um eine Objektinstanz handelt, empfiehlt sich das TYPO3 CMS Plugin. Darüber hinaus gibt einem das Plugin im Controller ein paar nützliche Hinweise.

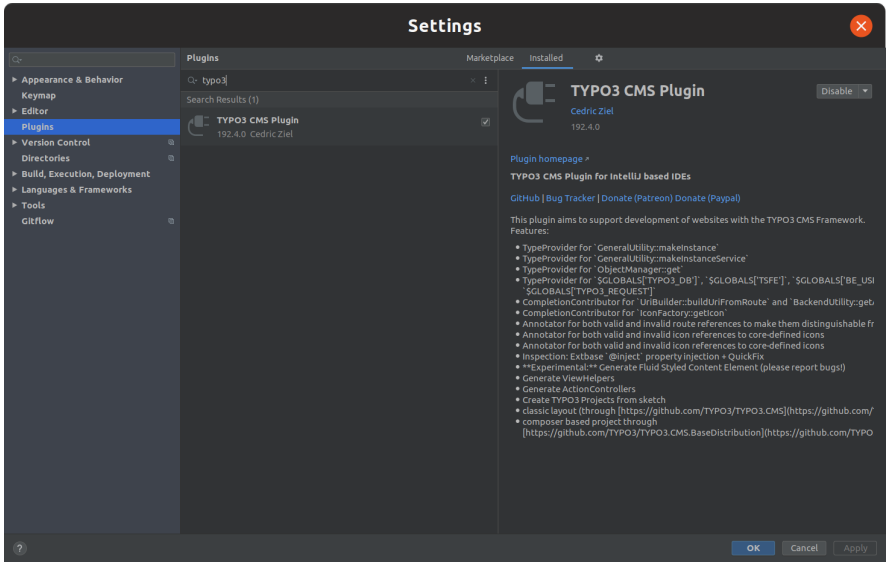


Abb: Installation des PhpStorm Plugins TYPO3 CMS

TYPO3 Coding Guidelines

TYPO3 empfiehlt sich an entsprechende Coding-Vorgaben zu halten: https://docs.typo3.org/m/typo3/reference-coreapi/master/en-us/CodingGuidelines/Index.html. Dank PSR-1/2 ist es sehr einfach, dies in PhpStorm voreinzustellen. Unter Settings/Editor/Code Style/PHP kann man „Set from“ und „PSR1/PSR2“ auswählen.

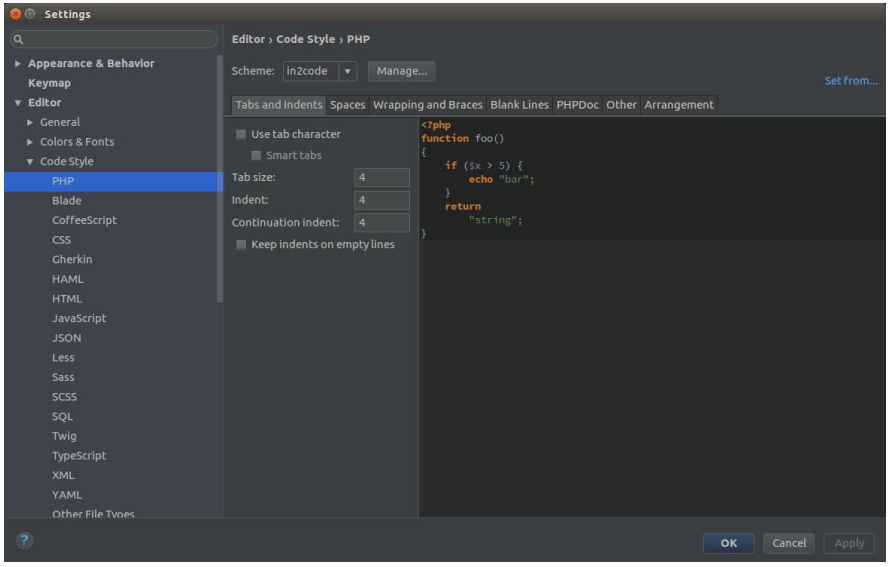


Abb: Einstellung der Coding Guidelines in PhpStorm

Tipp: Wenn man sich für den PHP Code Sniffer in PhpStorm entscheidet (siehe https://blog.jetbrains.com/phpstorm/2014/06/using-php-code-sniffer-in-phpstorm/) bekommt man bereits beim Schreiben entsprechende Warnhinweise.

TYPO3 Coding Guidelines

PhpStorm kann noch deutlich mehr – hier ein kleiner Auszug hilfreicher Funktionen:

- Ausführung von Unittests
- Unterstützung von XDEBUG
- Unterstützung von Versionierungssoftware wie GIT oder SVN
- Ausführung von NPM- oder Gulp-Tasks
- Ausführung von composer Befehlen
- Zugriff auf die lokale Datenbank
- PhpStorm Plugins erweitern die Funktionalität noch mehr (z.B. GIT Flow Plugin)

Extbase - Grundlegendes

Dateistruktur

Eine Extbase-Extension lässt sich sofort an der markanten Dateistruktur erkennen. Diese Datei- und Ordnerstruktur ist teilweise zwingend nötig. Neu ist die Schreibweise von Dateien innerhalb einer solchen Erweiterung. Dateinamen werden in UpperCamelCase geschrieben (siehe Dateien und deren Bedeutung).

Im Root-Verzeichnis einer Extbase-Extension befinden sich einige Dateien, die es auch in Abstract Plugin Extensions gibt. Alleine die Schreibweise (durchgehend mit lowerunderscored) deutet bereits darauf hin, dass es sich hierbei nicht um Extbase-spezifische Files handelt.

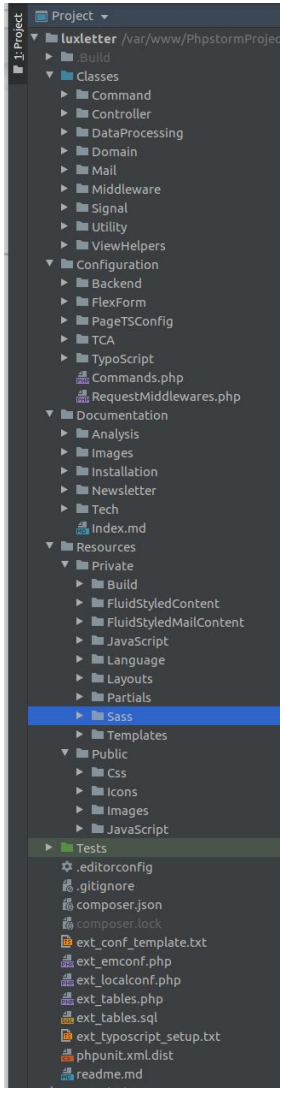


Abb: Beispiel Verzeichnisstruktur

Dateiname	Erklärung
composer.json	Soll die Extension über Composer geladen werden können, wird eine Konfiguration benötigt. **Hinweis:** Übrigens lassen hier auch gleich Drittkomponenten definieren, die bei der Installation mit geladen werden können.
ext_emconf.php	„emconf“ steht für Extension Manager Configuration Beim Öffnen fällt auf, dass hier lediglich ein Array mit Aussagen zur Extension zu finden ist. Diese Art der Aussagen werden vom Extension Manager für dessen Auflistung benötigt.
ext_localconf.php	Ähnlich wie die LocalConfiguration.php (ehemals localconf.php) befindet sich hier die Grundkonfiguration der Extension. Unter anderen werden hier Extbase und piBase Frontend-Plugins ins TYPO3 eingebunden.
ext_tables.php	Die Datei funktioniert als Pendant zu ext_localconf.php. Jedoch wird diese Konfiguration nur geladen, wenn zeitgleich im Backend eingeloggt ist. Backend-Icons oder TSConfig wird hier z.B. registriert.
ext_tables.sql	Diese Datei besteht aus einer Reihe von SQL-Anweisungen, die bei der Erstellung eigener Tabellen in der vorhandenen Datenbank wichtig sind. **Hinweis:** Obwohl über jeder Tabelle ein „CREATE TABLE…“ steht, werden Tabellen für TYPO3 nicht erneut angelegt, wenn diese bereits vorhanden sind. Updates sind natürlich
	weiterhin möglich.
Classes	Im Verzeichnis Classes befindet sich alle PHP-Dateien wie Controller, Model, sowie Repository (und unter Umständen weitere Funktionen wie Validatoren, eigene Klassen und ViewHelper)
Classes/Controller	In diesem Unterverzeichnis von Classes befinden sich die Controller-Dateien der Extension. Hierbei handelt es sich um das Herzstück der Extension.
Classes/Domain	In Domain sollte sich (basierend auf den Konzepten von MVC und DDD) die komplette Businesslogik der Anwendung befinden. Üblicherweise gibt es dort mindestens zwei Verzeichnisse (Repository und Model).
Classes/Utility	Der Utility-Ordner enthält üblicherweise Utility Klassen mit statischen Funktionen (wie z.B. Stringfunktionen, etc…)
Classes/ViewHelpers	Dort befinden sich die ViewHelper der Extension, die in Fluid verwendet werden können. **Hinweis:** ViewHelper aus einer Extension lassen sich auch im Seitenkontext oder in anderen Extensions verwenden
Configuration	In Configuration befindet sich die Konfiguration der Erweiterung. Üblicherweise also TCA, FlexForm, Routingdefinitionen, etc…
Documentation	Neben einer Readme.md Datei im Root-Verzeichnis befindet sich in diesem

	Ordner die ausführliche Dokumentation zur Extension (im Format .rst oder .md)
Resources	Hier werden Resourcen zur Extension abgelegt. Üblicherweise beinhaltet dieses Verzeichnis lediglich zwei Unterverzeichnisse (siehe nachfolgende Zeilen).
Resources/Private	Beinhaltet alle Resources, die über den Apache nicht zugreifbar sind (Sprachdateien, HTML-Templates, etc...).
Resources/Public	Das Pendant zu Private beinhaltet die Resources die über den Browser und den Apache aufrufbar sein müssen (Bilder, CSS, JavaScript, etc...).
Tests	Hier befinden sich alle Arten von automatisierten Tests, ganz gleich ob es sich um Unit-, Functional-, Behaviour- bzw. Acceptance- oder Visual-Regression-Tests handelt.

Tipp: Es gibt keine Einschränkungen neue Verzeichnisse oder Dateien anzulegen. Im Gegenteil raten wir sogar dazu, sich sinnvolle Strukturen zu überlegen und lieber viele kleine Dateien an Stelle weniger umfangreicher Dateien zu nutzen.

Datenstruktur

Alle, von der Extension angelegten Datenbank-Tabellen, folgen auch einer strikten Namenskonvention (sofern es sich um Objekte als Teil des ORM handelt):

tx_[extkey]_domain_model_[Objektname]

Abb: Beispielstruktur als Teil der News-Extension

Die erste Extension

“Kickstart” mit dem Extension Builder

Mit Hilfe der Erweiterung „extension_builder“ (siehe https://packagist.org/packages/friendsoftypo3/extension-builder) lässt sich sehr schnell eine Extension erstellen. Dies ist nicht zwingend nötig, erspart aber einiges an Tipparbeit. Die Extension lässt sich ber composer dazu laden und im Extension Manager (oder über die Konsole) aktivieren. Nach einem Reload des Backends steht den Administratoren ein gleichnamiges Modul zur Verfügung.

Wichtig wenn TYPO3 im Composer-Mode läuft, ist das richtige Autoloading der PHP-Dateien:

```
{
...
  "autoload": {
        "psr-4": {
        "In2code\\Persons\\":
"public/typo3conf/ext/persons/Classes"
        }
  },
}
```

Im Anschluss den Befehl „composer dump“ auf der Kommandozeile ausführen. Danach sollten unsere Dateien automatisch eingebunden werden.

Die Erweiterung mit den nebenstehenden Eigenschaften kann durch Klick auf das Speichern-Symbol erzeugt werden.

Erklärung: In der linken Spalte beschreiben wir die Extension im Allgemeinen und geben des Weiteren an, dass wir ein Frontend-Plugin benötigen. In der rechten Spalte lässt sich über Drag’n Drop ein Neues Objekt-Modell erzeugen.

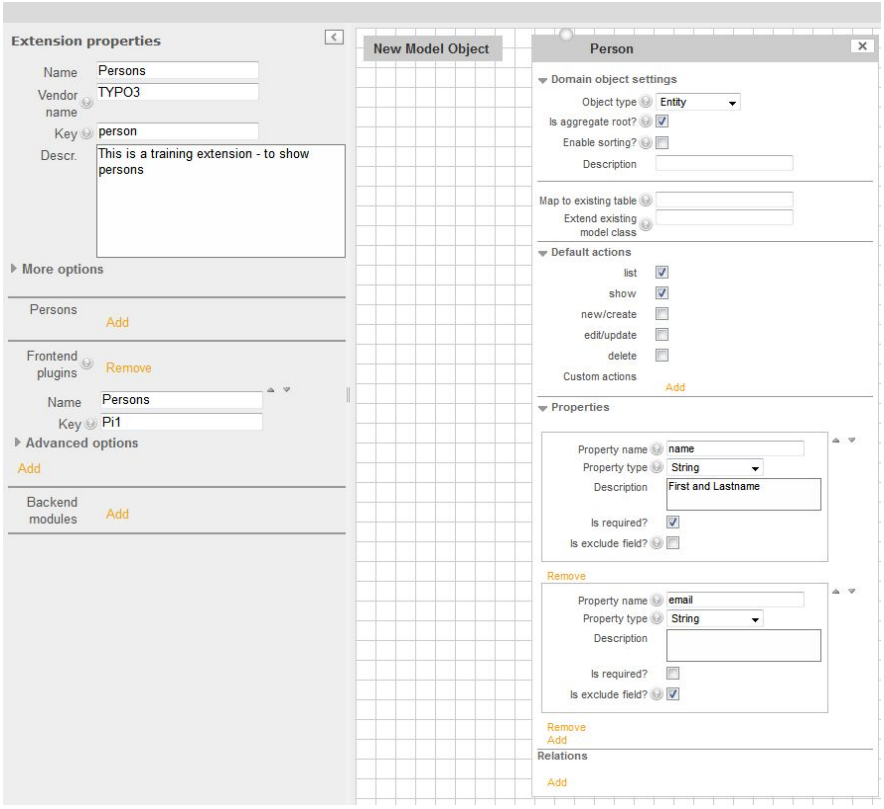


Abb: Beispiel Erweiterung mit dem Extension Key “person”

Tipp: Beim Modellieren sind einige Felder so lange nicht sichtbar bis man rechts oben auf “Show advanced options” klickt

Die erste Ausgabe

Datensätze anlegen

Nach dem Speichern der neuen Extension “Persons” über das Install Tool oder die Konsole einen Database-Compare machen, damit neue Tabellen angelegt werden.

Im Prinzip steht die neue Erweiterung bereits in den Startlöchern und kann auch schon Datensätze aus der Datenbank anzeigen.

Im Backend können nun Seiten und Datensätze (siehe nachfolgende Bilder) angelegt werden.

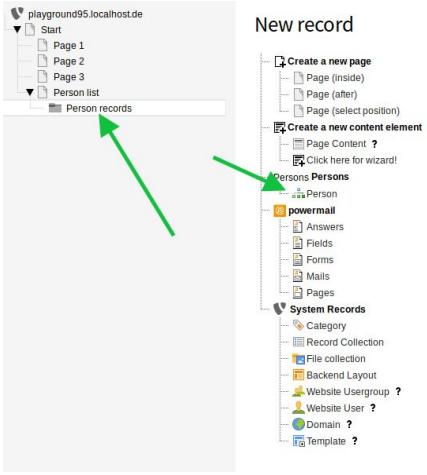


Abb: Beispiel-Seitenstruktur und Beispiel-Datensätze

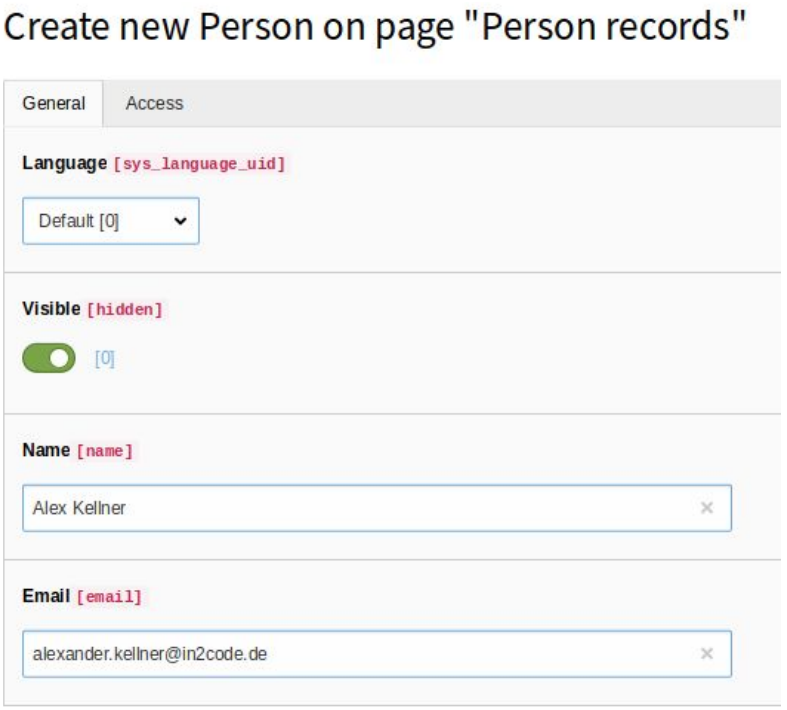


Abb: Datensatz erzeugen und abspeichern

Plugin anlegen

Auf der Seite „Person list“ kann nun ein neuer Seiteninhalt vom Typ Plugin eingefügt werden. Im Anschluss das Plugin Persons auswählen.

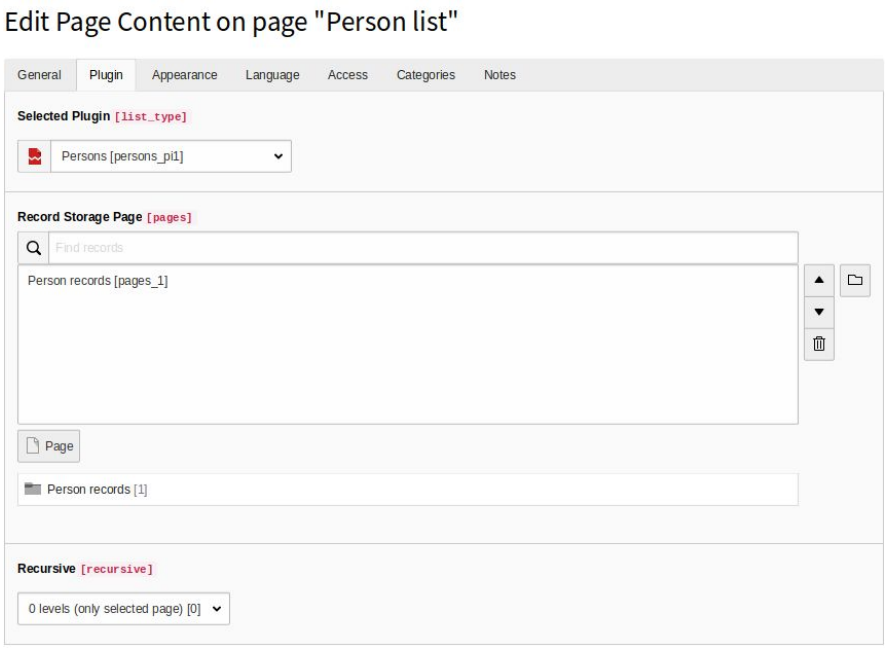


Abb: Plugin Ansicht

Wichtig ist außerdem den Ausgangsort mit den Datensätzen zu selektieren, sonst erhält man später keine Ausgabe.

Ausgabe im Frontend

Abb: Listenansicht im Frontend

Abb: Detailansicht im Frontend

Tipp: Eventuell muss der Frontend- oder Systemcache von TYPO3 einmal geleert werden, wenn es zu Problemen an dieser Stelle kommt.

Extension - Programmierung

Alle nachfolgenden Codebeispiele oder Screenshots beziehen sich auf die zuvor erzeugte Extension “Persons”

Einbindung des Plugins

Ein weiterer Vorteil bei der Benutzung eines Plugins mit Extbase ist die genaue Einstellung, welche Ansicht gecacht werden soll und welche nicht. In der Regel gilt, dass man jede Ansicht aus Performancegründen cachen sollte, in bestimmten Fällen oder in der Entwicklung kann es sinnvoll sein, einzelne Views hiervon auszunehmen. In der Regel sind dies Views, die eine Interaktion des Benutzers ermöglichen (Edit, New, Filter, etc...).

Beim Öffnen der Datei ext_localconf.php fällt uns die Einbindung des FE-Plugins auf. Der Methode configurePlugin() werden 4 Parameter übergeben:

- Die Extension-Bezeichnung (Vendor.Extensionkey)
- Der Name des Plugins (in der Regel Pi1, Pi2, Pi3 – Pi steht hierbei historisch für Plugin)
- Ein Array bestehend aus allen Controllern mit dessen Actions (Views)
- Ein Array bestehend aus allen Controllern mit dessen Actions (Views) genau wie unter Punkt 3 mit dem Unterschied, dass jede angegebene Action nicht gecacht wird.

In der Regel wird die erste Action im ersten Controller aufgerufen, sollte im Frontend nichts anderes angegeben werden.

```
<?php
…

\TYPO3\CMS\Extbase\Utility\ExtensionUtility::configurePlugin(
        'In2code.Persons',
        'Pi1',
        [
        'Person' => 'list, show'
        ],
        // non-cacheable actions
        [
        'Person' => ''
        ]
);
…
```

Der Controller

Der Controller besteht aus zwei Methoden (zwei Views oder auch zwei Actions) sowie einem Attribut. Jede beschriebene Action in der ext_localconf.php findet sich im entsprechenden Controller unter [actionName]Action() wieder.

Somit gibt es eine Listen- (listAction) und eine Detailansicht (showAction).

```
<?php
namespace In2code\Persons\Controller;

/**
 * PersonController
 */
class PersonController extends
\TYPO3\CMS\Extbase\Mvc\Controller\ActionController
{

        /**
        * @var
\In2code\Persons\Domain\Repository\PersonRepository
        * @inject
        */
        protected $personRepository = null;

        /**
        * action list
        *
        * @return void
        */
        public function listAction()
        {
        $persons = $this->personRepository->findAll();
        $this->view->assign('persons', $persons);
        }

        /**
        * action show
        *
        * @param \In2code\Persons\Domain\Model\Person $person
        * @return void
        */
        public function
showAction(\In2code\Persons\Domain\Model\Person $person)
        {
        $this->view->assign('person', $person);
        }
}
```

Standardmäßig wird immer der erste definierte Controller mit der ersten definierten Action (siehe ext_localconf.php) aufgerufen. Ein Umschalten lässt sich über GET-Parameter beim Aufruf der Seite erzwingen:
index.php?id=1&tx_persons_pi1[action]=actionName&tx_persons_pi1[controller]=Controllername

Tipp: Wenn man Actions ohne den cHash Parameter aufruft (z.B. händisch zum Test oder via AJAX) muss man das explizit via TypoScript Setup erlauben:
plugin.tx_persons_pi1.features.requireCHashArgumentForActionArguments = 0

Tipp: Das Konzept des **SlimControllers** sieht vor, die Methoden im Controller nicht mit unnötigen Anweisungen zu überladen. So gehören Dinge wie Businesslogik, Datenbankabfragen, Schnittstellenkommunikation, Vorbereitung der Konfiguration, etc... nicht in den Controller und eher in den Domain-Bereich. Eine gute Kennzahl sind maximal 8-10 Zeilen Code pro Action und nicht mehr als 10 Methoden insgesamt.

Methode listAction()

In der Methode listAction() wird auf alle vorhandenen Datensätze abgefragt (mit Hilfe von findAll() aus dem Repository). In der nächsten Zeile werden diese Datensätze an den View (also Fluid) übergeben.

Da es sich bei der listAction um den ersten View in der ext_localconf.php handelt, wird diese auch im Frontend angezeigt (sofern keine weiteren Parameter übergeben werden)

Methode showAction()

Die Methode unterscheidet sich zur listAction vor allem durch die Übergabe eines Parameters $person.

In der Funktion wird lediglich dieser Personendatensatz wieder an den View übergeben.

Möchte man den Detailview im Frontend aufrufen, muss man das Extbase entsprechend mit dem GET-Paramter in der URL ***&tx_persons_pi1[action]=show*** mitteilen

Tipp: Beim Aufruf der URL mit ***&tx_persons_pi1[action]=show*** erscheint ein entsprechender Fehler „An error occurred while trying to call TYPO3\Person\Controller\PersonController->showAction(). Error: Required property 'person' does not exist.“ Das liegt daran, dass die Methode showAction() den Parameter $person erwartet. Der korrekte Aufruf zur Anzeige des Datensatz mit der UID 1 lautet also ***&tx_persons_pi1[action]=show&tx_persons_pi1[person]=1***

Attribut $personRepository

Mit Hilfe von Dependendcy Injection (siehe http://de.wikipedia.org/wiki/Dependency_Injection) und dem Zusatz @inject im Kommentar oberhalb wird das Object $this->personRepository in der kompletten Klasse zur Verfügung gestellt. Somit funktioniert auch die Datenabfrage in der listAction().

Tipp: Der Einsatz von Dependency Injection im Controller ist nützlich wenn man ein Objekte in mehreren Methoden benötigt, da dies durch wenige Zeilen geschieht. Dies hat jedoch negative Auswirkungen auf die Performance (siehe https://wiki.typo3.org/Dependency_Injection)

Neben der Möglichkeit der Nutzung von Dependency Injection, können Repositories genauso über einen Konstruktor (__construct()) für die ganze Klasse zur Verfügung gestellt werden.

Des Weiteren kann man ein Repository auch lokal innerhalb einer Funktion instanzieren (mit dem Object Manager über $this->objectManager->get(MyRespository::class)). Das ist eine gute Möglichkeit, wenn man das Repository nur in einer Methode benötigt.

Nützliche Methoden im Controller

Nachfolgend ein paar Beispiele wichtiger Methoden, die man immer wieder im Controller benötigt. Einige Methoden lassen sich auch im Model, Repository oder in einem ViewHelper nutzen.

PHP	Erklärung
$this->view->assign('name', 'wert');	Übergabe eines Wertes (z.B. String, Array oder Objekt) an Fluid
$array = ['name' => 'wert', 'name2' => 'wert2']; $this->view->assignMultiple ($array);	Mehrere Werte an Fluid übergeben. Aufruf in Fluid dann im Anschluss mit {name} oder {name2}
$this->view-> setTemplatePathAndFilename();	Anpassen des Templatepfades (in der Regel nicht nötig, da über TypoScript möglich) für außergewöhnliche Fälle (z.B. E-Mail Template)
$string = $this->view->render();	Soll der Rückgabewert aus dem Fluidtemplate nicht direkt an TYPO3 übergeben werden, kann man auch in eine Variable rendern.
$this->request->getArguments();	Abfrage auf alle Pluginvariablen (GET oder POST Parameter innerhalb von tx_persons_pi1)
$this->request->getArgument('action');	Abfrage auf ein bestimmtes Argument (GET oder POST Parameter innerhalb von tx_persons_pi1)
$this->addFlashmessage('gespeichert!');	Hiermit können Flashmessages für die Ausgabe im Fronend gespeichert werden.
$this->forward('actionName');	Weiterleitung auf eine andere Action (ohne Browserreload). Parameter: 1. Actionname 2. Controllername (leer: Aktueller Controller) 3. Extensionname (leer: Aktuelle Extension) 4. Weitere Argumente als array
$this->redirect('actionName');	Weiterleitung auf eine andere Action (mit Browserreload). Gleiche Parameter wie $this->forward();
$this->configurationManager->getContentObject();	Hiermit erhält man das zum Plugin passende ContentObject. Hilfreich um an Werte des aktuellen tt_content Datensatz zu gelangen.
\TYPO3\CMS\Extbase\ Utility\LocalizationUtility:: translate('key', $this->extensionName)	Aufruf eines Wertes aus locallang.xlf (**Hinweis**: Dies hat eigentlich nichts mit dem Controller zu tun. Die statische Funktion kann von überall aus aufgerufen werden).
initializeAction()	Bei Bedarf kann neben den normalen Actions eine initializeAction (ähnlich einem Constructor) eingebaut werden.

initializeShowAction()	Ähnliche wie die initializeAction() gibt es die Möglichkeit eine Funktion vor dem Aufruf der eigentlichen Action zu schalten (in diesem Fall die showAction()). Übliche Anwendung zur Prüfung der Autorisierung oder zum Ändern der Parameter.
$this->objectManager->get('ClassName');	Eine Klasse als Objekt instanzieren. Hinweis: In TYPO3 sollten Klassen immer über den ObjectManager oder über GeneralUtility::makeInstance() instanziert werden. Somit wird beispielsweise Xclassing (also das Überschreiben von Klassen) oder der Einsatz des SingeltonInterface erst möglich.

Nützliche Eigenschaften im Controller

PHP	Erklärung
$this->settings	Konfigurationswerte aus dem TypoScript oder aus dem Flexform als array. **Hinweis**: Dies ist die einzige Variable nicht nicht extra an Fluid übergeben werden muss. Über {settings} gelangt man automatisch in Fluid an die Parameter.
$this->extensionName	Mehrere Werte an Fluid übergeben. Aufruf in Fluid dann im Anschluss mit {name} oder {name2}
$this->actionMethodName	Aktuelle Action

Der View

Per Konvention ist festgelegt, dass bei Aufruf der listAction() im PersonController ein entsprechendes HTML-Template geladen wird. Extbase sucht die Datei automatisch entsprechend unter diesem Pfad:

Resources/Private/Templates/Person/List.html

Wird die Detailansicht im Frontend aufgerufen, zieht die showAction() entsprechend ein Template nach sich:

Resources/Private/Templates/Person/Show.html

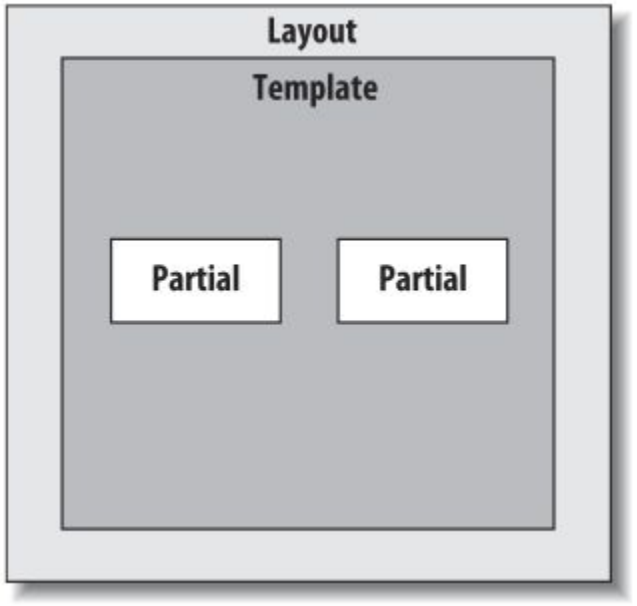


Abb: Zeigt Aufteilung von Templates, Partials und Layouts in Fluid

Aufteilung im Template

Vereinfachte Darstellung:

```
<f:layout name="Default" />

Interne Notizen

<f:section name="main">
   HTML für Ausgabe
</f:section>
```

Der Extension Builder hat das HTML-Template bereits so angelegt, dass ein Layout aufgerufen werden soll. Der Name Default weißt auf folgende Pfad/Datei Kombination:
Resources/Private/Layouts/Default.html

Die Layout Datei beinhaltet drei Zeilen und weist Extbase an, wieder zum Template zurückzuspringen und lediglich den Teil zwischen <f:section name="main"> und </f:section> zu rendern.

```
<div class="tx-persons">
    <f:render section="main" />
</div>
```

Parameterübergabe vom Controller an Fluid

3 verschiedene Möglichkeiten stehen zur Verfügung, Werte mit der Methode assign() zu übergeben:

Übergabe Wert	Beispiel	Aufruf in Fluid	Erklärung
String	$this->view->assign ('value', 'foo');	{value}	Bei der Übergabe eines Strings kann man diesen direkt wieder in Fluid aufgreifen
Array	$this->view->assign ('value', [0 => 'foo']);	{value.0}	Bei der Übergabe eines Arrays, kann man durch Trennung mit einem Punkt den entsprechenden Key nutzen
Object	$person = $this->personRepository->findByUid(1); $this->view->assign ('person', $person);	{person.name}	So bald ein Objekt mit Getter und Setter bereitsteht, lässt sich dieses ebenfalls an den View übergeben. Wenn hier also mit .name auf einen Wert zugegriffen wird, gibt es sehr wahrscheinlich eine Methode getName(). **Hinweis**: Auch auf Eigenschaften aus der Klasse oder auf getter, die mit “is” anfangen, kann zugegriffen

			werden. **Hinweis**: Zugreifen kann Fluid nur, wenn es sich um Public Methoden oder Eigenschaften handelt.

Das Rendering im Template

```
<table  class="tx_person" >
    <tr>
        <th><f:translate
key="tx_person_domain_model_person.name" /></th>
        <th><f:translate
key="tx_person_domain_model_person.email" /></th>
        <th> </th>
        <th> </th>
    </tr>

    <f:for each="{persons}" as="person">
        <tr>
                <td><f:link.action action="show"
arguments="{person : person}">
{person.name}</f:link.action></td>
                <td><f:link.action action="show"
arguments="{person : person}">
{person.email}</f:link.action></td>
                <td><f:link.action action="edit"
arguments="{person : person}">Edit</f:link.action></td>
                <td><f:link.action action="delete"
arguments="{person : person}">Delete</f:link.action></td>
        </tr>
    </f:for>
</table>

<f:link.action action="new">New Person</f:link.action>
</f:section>
```

Wenn man sich nun den Bereich zwischen den Section-Tags ansieht, erkennt man einen Mix aus HTML-Templates und Fluid-Anweisungen, die wie HTML-Tags aussehen – hierbei handelt es sich um ViewHelper.

Ein ViewHelper ist, wie der Name schon sagt, eine Unterstützung bei der Ausgabe von HTML-Code.

Oben stehend erkennt man beispielsweise einen Translate-ViewHelper. Dieser gibt einen lokalisierten (übersetzten) Text aus einer Übersetzungsdaten (locallang.xlf oder locallang.xml) je nach Frontend-Sprache aus.

Bei dem folgenden ForEach-ViewHelper handelt es sich um eine Schleife – das bedeutet, dass sämtlicher HTML-Code zwischen <f:for> und </f:for> so lange wiederholt wird, wie es Objekte in der Variable {persons} gibt. Zugleich stellt der ViewHelper eine neue Variable in der Schleife zur Verfügung {person}. Die Funktionsweise verhält sich also ähnlich zur PHP-Funktion foreach.

Innerhalb der Schleife lassen sich die Felder aus der Personentabelle mit {objekt.feldname} wieder ausgeben (z.B. {person.email})

Einen weiteren ViewHelper kann man um jedes Feld in der Schleife erkennen: Der Extension Builder hat jeden Zelleninhalt mit einem Link versehen <f:link.action></f:link.action>. Hierbei lässt sich auf einen anderen View verlinken und bereits Argumente mitgeben.

Das Repository

```
namespace TYPO3\Persons\Domain\Repository;

/*
    Einige Kommentare
*/
class PersonRepository extends
\TYPO3\CMS\Extbase\Persistence\Repository
{
}
```

Über das Repository lassen sich Datenbank-Abfragen gestalten.

Der Aufruf $this->personRepository->findAll() im Controller bemüht das PersonRepository um eine Methode um alle Objekte zu finden.

Etwas befremdlich ist der erste Anblick der Datei PersonRepository.php, die vollkommen leer zu sein scheint.

Vermutlich wird der eine oder andere hier eine Funktion mit Namen findAll() erwartet haben, diese ist jedoch nicht zu sehen. Lediglich eine Klasse die wiederum erweitert wird ist sichtbar.

Die Extbase Klasse \TYPO3\CMS\Extbase\Persistence\Repository beinhaltet bereits die wichtigsten Repository-Abfragen, die wir nicht noch einmal aufnehmen müssen.

Methoden im Repository

Methode	Erklärung	Funktion
findAll()	Sucht alle Objekte innerhalb der Datenbank. **Hinweis**: Funktioniert nur wenn der Ausgangspunkt im Plugin oder per TypoScript gesetzt wurde	lesend
findByUid(123)	Findet einen Datensatz an Hand der Uid	lesend
findBy[Feldname]('foo')	Es stehen automatisch Methoden zu jedem Feld bereit. Gibt es beispielsweise ein Feld namens „email“, gibt es auch die Methode findByEmail(). **Hinweis**: So schön das im ersten Moment klingt, kann so eine Methode nur genutzt werden, wenn der String eindeutig ist. Soll auch mit einem Teilstring gesucht werden, muss man selbst Hand an legen.	lesend
findOneBy[Feldname]('foo')	Die Funktionsweise ist ähnlich wie bei findBy[Feldname](), jedoch wird hier nur ein Objekt zurückgegeben (limit=1)	lesend
countAll()	Anzahl aller Objekte	lesend
countBy[Feldname]('foo')	Anzahl aller Objekte, die in einem bestimmten Feld einen Wert enthalten	lesend
add($object)	Ein Objekt wird dem Repository hinzugefügt (ein neuer Datensatz wird erstellt)	schreibend
remove($object)	Ein Objekt wird aus dem Repository entfernt (ein Datensatz wird aus der Tabelle gelöscht)	schreibend
removeAll()	Löscht alle Datensätze	schreibend
replace($obj1, $obj2)	Ersetzt ein Objekt durch ein anderes	schreibend
update($object)	Die Änderungen an einem Datensatz werden gespeichert	schreibend
insert($object)	Dieses Objekt wird als Datensatz neu gespeichert	schreibend

Eigene Methoden nutzen

Eigene Methoden können dem Repository hinzugefügt werden. Im ersten Beispiel wollen wir alle Personen finden, diese aber sortiert zurück geben.

```
/**
 * Find all persons
 *
 * @return query object
 */
public function getAllPersons()
{
    $query = $this->createQuery();
    $query->setOrderings(array('name' =>
\TYPO3\CMS\Extbase\Persistence\QueryInterface::ORDER_DESCENDING)
);
    return $query->execute();
}
```

Die Nutzung von createQuery() liefert bereits ein QueryObject mit Standardeinstellungen (Ignoriere deleted=1, hidden=1, Start- und Stoppzeiten, etc...). Will man diese Überschreiben, ist dies über eigene Funktionen möglich.

Aufruf der Methode getAllPersons() im Controller:

```
public function listAction()
{
    $persons = $this->personRepository->getAllPersons();
    $this->view->assign('persons', $persons);
}
```

Tipp: Natürlich kann man auch vorhandene Funktionen überschreiben (überladen). Dies empfiehlt sich jedoch nur in den seltensten Fällen. Wenn Standardfunktionen plötzlich ein anderes Verhalten zeigen, führt dies bei Kollegen eventuell zu Verwirrung und erhöht die Dauer einer potenziellen Fehlersuche.

In einem zweiten Beispiel, wird bereits aus dem Controller ein Suchparameter übergeben, nach dem die Abfrage im Repository filtern soll.

```
/**
 * Find all persons with given searchterm
 *
 * @param string $searchparam
 * @return query object
 */
public function getAllPersons(string $searchterm)
{
    $query = $this->createQuery();

    $or = [
        $query->like('email', '%' . $searchterm . '%'),
        $query->like('name', '%' . $searchterm . '%')
    ];
    $query->matching($query->logicalOr($or));

    $result = $query->execute();
    return $result;
}
```

Methoden im Überblick

Methode	Erklärung
$this->createQuery()	Erzeugt eine neue Query. Steht in der Regel am Anfang einer Repository-Funktion.
logialAnd ($constraints)	Verbindet einzelne Abfrage mit der logischen Funktion UND
logialOr ($constraints)	Verbindet einzelne Abfrage mit der logischen Funktion ODER
logialNot ($constraints)	Logische Funktion NOT zur Negierung
setOrderings()	Ändert die Standardsortierung
setLimit()	Setzt ein Limit für die Abfrage (ein Integer-Wert wird erwartet)
setOffset()	Setzt einen Versatz – sinnvoll z.B. für einen eigenen Pagebrowser
count()	Zählt die Objekte
getFirst()	Gibt den ersten Datensatz zurück
execute()	Führt den Query aus. Steht in der Regel am Ende jeder Repository-Funktion.

Eigene SQL-Abfragen

Sollten die vorhandenen Methoden nicht ausreichen, ein bestimmtes SQL-Query zu erzeugen, lässt sich das auch selber machen.

Die Gründe warum man ein SQL-Query nicht selber ausführen sollte, liegen vor allem in der Flexibilität der Anwendung. Sollte in Zukunft auf eine SQL-Datenbank (nosql oder textdatei, etc...) verzichtet werden, funktioniert die eigene Abfrage nicht mehr.

```
public function getPageTitleByUid($uid)
{
    $query = $this->createQuery();

    $sql = 'select title';
    $sql .= ' from pages';
    $sql .= ' where uid = ' . intval($uid);
    $sql .= ' limit 1';

    $query->getQuerySettings()->setReturnRawQueryResult(true);
    $result = $query->statement($sql)->execute();
    return $result[0]['title'];
}
```

Das Model

Im Model wird der Datensatz mit allen Eigenschaften beschrieben.

Wie im Absatz OOP erklärt, besteht ein Model nicht ausschließlich aus Attributen, über die man von außen zugreifen kann, sondern aus einem (oder mehreren) Attributen mit eigenen Methoden, diese auszulesen oder zu verändern (Getter und Setter).

Sobald eine Methode getName() im Model verfügbar ist, lässt sich über Fluid mit {object.name} auf das gleichnamige Attribut zugreifen.

```
<?php
namespace In2code\Persons\Domain\Model;

/**
 * Person
 */
class Person extends
\TYPO3\CMS\Extbase\DomainObject\AbstractEntity
{
	/**
	 * @var string
	 * @TYPO3\CMS\Extbase\Annotation\Validate("NotEmpty")
	 */
	protected $name = '';

	/**
	 * @var string
	 */
	protected $email = '';

	/**
	 * @return string $name
	 */
	public function getName()
	{
		return $this->name;
	}

	/**
	 * @param string $name
	 * @return void
	 */
	public function setName($name)
	{
		$this->name = $name;
	}

	/**
	 * @return string $email
	 */
	public function getEmail()
	{
		return $this->email;
	}

	/**
	 * @param string $email
	 * @return void
	 */
	public function setEmail($email)
	{
		$this->email = $email;
	}
}
```

Über die Dokumentation (PhpDoc) oberhalb der Attribute und Methoden wird auch die Validierung der Werte bestimmt. Sollte also im Modell ein Attribut mit bsp. NotEmpty gekennzeichnet sein, ist es nicht möglich im Frontend Datensätze mit leeren Feldern zu zeigen. Dies führt unweigerlich zu Fehlermeldungen.

Vermeidung eines AnemicDomainModel

Wie eingangs erwähnt, sollte sich die Businesslogik im Model oder zumindest im Bereich Domain befinden. In diesem Beispiel sieht man jedoch lediglich ein “dummes” Model ohne Logik. Hierbei spricht man von einem AnemicDomainModel oder AnemicModel. Es gilt dieses Antipattern zu vermeiden.

Ein einfaches Beispiel für die Unterbringen von Logik im Model wäre, einen weiteren Getter hinzuzufügen, der beispielsweise zurückgibt, ob eine Person bereits über 18 Jahre alt ist oder noch nicht. Weiter wäre es möglich, die allgemeine Schreibweise von “Nachname, Vorname” auch im Model aufzunehmen, wenn dies im Frontend immer wieder benötigt wird.

```
/**
 * @var string
 * @validate NotEmpty
 */
protected $lastName = '';

/**
 * @var string
 * @validate NotEmpty
 */
protected $firstName = '';

/**
 * @var \DateTime
 */
protected $birthDate = null;

/**
 * @return string $lastName
 */
public function getLastName(): string
{
    return $this->lastName;
}

/**
 * @return string $firstName
 */
public function getFirstName(): string
{
    return $this->firstName;
}

/**
 * @return \DateTime $birthDate
 */
public function getBirthDate(): ?\DateTime
{
    return $this->birthDate;
}

/**
 * @return string
 */
public function getFullName(): string
{
    return $this->getLastName() . ', ' . $this->getFirstName();
}

/**
 * @return bool
 */
public function isAtLeast18(): bool
{
    $birthdate = $this->getBirthDate();
    return $birthdate->diff(new \DateTime())->y >= 18;
}
```

Tipp: Der Aufruf im Fluid wäre {person.atLeast18} bzw. {person.fullName}. **Hinweis**: Diese Getter sieht man übrigens nicht bei einem f:debug auf das Objekt, da dieser var_dump lediglich alle Eigenschaften aus dem Model anzeigt.

Fluid - der View

Fluid ist eine mächtige Template-Engine die das veraltete TEMPLATE cObject von TYPO3 vollständig ersetzt und um neue Funktionen und Eigenschaften ergänzt.

Wie bereits oben erwähnt, kann Fluid auch im Kontext einer Extension eingesetzt werden.

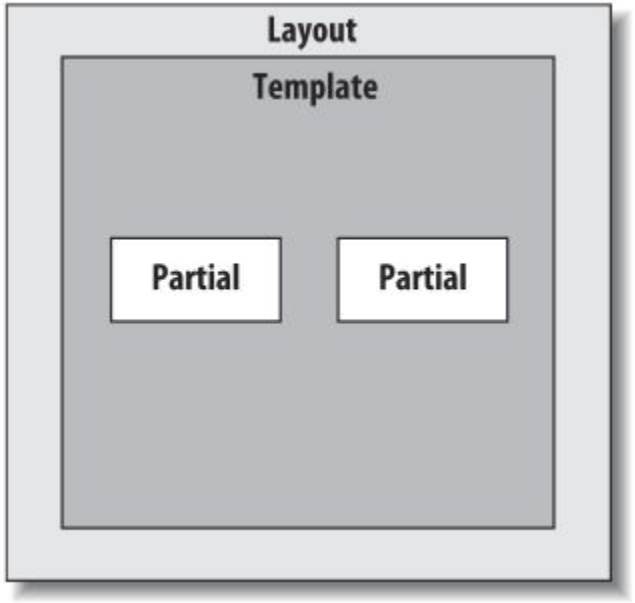


Abb: Zeigt Aufteilung von Templates, Partials und Layouts in Fluid

Aufteilung

Aktuell gibt es verschiedene Möglichkeiten, wie man das Template aufteilen kann. Eine Aufteilung hilft, einfacher Skalierbar zu werden. Einige Möglichkeiten benötigen eigene Dateien, andere können in der gleichen Datei Verwendung finden.

Name	Erklärung
Templates	Template ist das Haupt-HTML-Template. Dieses wird immer zuerst aufgerufen und beinhaltet in der Regel den Hauptteil der Ausgabe.
Partials	Ein Partial ist eine Art Snippet. Hierbei handelt es sich also um einen ausgelagerten Teilbereich. Der Vorteil des Auslagerns ist die Wiederverwendbarkeit. Wenn man beispielsweise eine Suchmaske in ein Partial auslagert, lässt sich diese in verschiedenen Templates wiederverwenden Partials können von Layouts, Templates und anderen Partials eingebunden werden.
Layouts	Bei einem Layout handelt es sich um das Gerüst der Extension. Im Frontend beinhaltet das Layout meist nur einen umschließenden DIV-Container mit einer Extensionklasse. Die Notwendigkeit des Anlegens weiterer Layouts ergeben sich, wenn man Views benötigt, die eventuell keinen oder andere DIV-Container benötigen (AJAX-Requests, XML, RSS, etc...).
Sections	Anders als bei den oberen Beispielen sind Sections Bereiche innerhalb einer Datei.

Beispielaufruf einer Section innerhalb eines Templates:

```
...
<f:render section="Video" arguments="{_all}"/>
...

<f:section name="Video">
   Video Content
</f:section>
```

ViewHelper

Das Grundkonzept der ViewHelper besteht darin, die Ausgabe um weitere Funktionen und Logiken anzureichern. Im Idealfall sieht ein ViewHelper auch aus wie ein HTML-Tag. Dadurch wird die IDE eine korrekte Einrückung vornehmen. Bei direktem Aufruf einer FLUID-Datei mit dem Browser, werden die ViewHelper ignoriert.

Vorhandene ViewHelper in Fluid und TYPO3:

ViewHelper	Erklärung
<f:count subject="{array}" />	Gibt die Anzahl der Objekte oder Inhalte aus einem Array wieder
<f:if condition="{var} > 3"> // do something </f:if> <f:if condition="{var}"> <f:then>A</f:then> <f:else>B</f:then> </f:if>	Ein Bereich wird im Fluid nur geparst, wenn die Condition zutrifft. Hinweis: Ein Stringvergleich ist nicht möglich – ein Workarround ist der Vergleich zweier Arrays
<f:for each="{objects}" as="object"> {object.title} </f:for> <f:for each="{objects}" as="object" key="number" iteration="itemIteration"> ... </f:for>	Foreach Schleife generiert eine neue Variable für den innenliegenden Bereich.
<f:debug>{object}</f:debug>	var_dump Debug auf String, Array oder

	Objekt. **Tipp**: <f:debug>{_all}</f:debug> zeigt alle zur Verfügung stehenden Variablen.
<f:format.crop maxCharacters="17" append=" [...]">This is some very long text</f:format.crop>	Beschneidet einen String auf einen bestimmten Wert. Hierbei werden vorhandene HTML-Tags nicht zerstört und bestehen weiterhin.
<f:format.currency currencySign="€" decimalSeparator="," thousandsSeparator="."> 1234.56</f:format.currency>	Formatierte Ausgabe mit Währung – Output 1.234,56 €
<f:format.number decimals="1" decimalSeparator="," thousandsSeparator="."> 2345.678</f:format.number>	Formatierte Ausgabe einer Zahl – Output 2.345,6
<f:format.date format="d.m.Y"> {date}</f:format.date> <f:format.date format="d.m.Y"> @{timestamp}</f:format.date>	Formatiertes Datum
<f:image src="fileadmin/image.png" width="200 alt="My Image" /> <f:image src="{f:uri.resource (path:'Images/Image.png')} " width="200 alt="My Image" />	Ausgabe eines Bildes ähnlich der Ausgabe mit TypoScript Unteres Beispiel mit Bild aus Verzeichnis Resources/Public/Images/Image.png
<f:link.action action="edit"> Bearbeiten</f:link.action> <f:uri.action action="edit" />	Link auf Action URL zu einer Action
<f:link.email email="foo@example.com" />	Link auf E-Mail
<f:link.external uri="http://www.typo3.org" target="_blank">typo3.org </f:link.external>	Externer Link
<f:link.page pageUid="23"> Contact</f:link.page> <f:uri.page pageUid="23" />	Interner Link URL zu einem internen Link
<f:link.typolink parameter=”123”>Link</f:link>	TypoLink zu einem Parameter. In diesem Beispiel zur Seite mit der UID 123.
<f:translate key="key" />	Label aus der Sprachdatei mit Berücksichtigung der aktuell eingestellten FE-Sprache (locallang.xlf oder locallang.xml)

Alle verfügbaren ViewHelper finden sich in der Dokumentation von TYPO3:
https://docs.typo3.org/other/typo3/view-helper-reference/9.5/en-us/

Outline und Inline Schreibweisen

Alle ViewHelper kann man in zwei verschiedenen Schreibweisen nutzen. Während die Outline-Schreibweise von der IDE als HTML-Tag beachtet und damit auch eingerückt wird, lässt sich die Inline-Schreibweise gut nutzen, wenn ViewHelper verschachtelt werden müssen:

```
Outline:
<f:translate key="idFromXml" />

Inline:
{f:translate(key:'idFromXml')}
```

```
ViewHelper mit Beginnendem und schließendem Tag
Outline:
<f:format.date format="d.m.Y">{date}</f:format.date>

Inline:
{date -> f:format.date(format:'d.m.Y')}
```

```
Beispiel für Verschachtelung
<f:image src="fileadmin/bild.jpg"
alt="{f:translate(key:'bildText')}" />
```

Tipp: Auf http://www.fluid-converter.com/ lassen sich Outline-ViewHelper automatisch zu Inline-ViewHelpern umwandeln.

Eigene ViewHelper

In einigen Fällen benötigt man mehr Funktionen im View als es bereits ViewHelper von Fluid gibt. In so einem Fall kann man etwas PHP nutzen um einen eigenen ViewHelper zu erstellen.

In nachfolgendem Beispiel wollen wir in der Lage sein, den ersten Buchstaben eines Strings großzuschreiben. Hierzu benötigen wir eine Datei mit dem Namen UcfirstViewHelper.php im Verzeichnis Classes/ViewHelpers/

```
<?php
declare(strict_types=1);
namespace In2code\Persons\ViewHelpers;

use TYPO3Fluid\Fluid\Core\ViewHelper\AbstractViewHelper;

/**
 * Class UcfirstViewHelper
 */
class UcfirstViewHelper extends AbstractViewHelper
{
        /**
         * @return void
         */
        public function initializeArguments()
        {
                parent::initializeArguments();
                $this->registerArgument('value', 'string', 'Any
string to convert', true);
        }

        /**
         * @return string
         */
        public function render(): string
        {
                return ucfirst($this->arguments['value']);
        }
}
```

Wichtig bei der Verwendung eigener ViewHelper, ist die Deklaration des eigenen Namespaces. Dies kann auf zwei unterschiedliche Wege funktionieren:

```
{namespace persons=TYPO3\Persons\ViewHelpers}
<f:layout name="Default" />

  Kommentar

<f:section name="main">
  <persons:ucfirst value="alex" />
</f:section>
```

```
<html xmlns:f="http://typo3.org/ns/TYPO3/CMS/Fluid/ViewHelpers"
xmlns:persons="http://typo3.org/ns/In2code/Persons/ViewHelpers"
data-namespace-typo3-fluid="true">

   <f:layout name="Default" />

      Kommentar

   <f:section name="main">
      <persons:ucfirst value="alex" />
   </f:section>

</html>
```

Der Vorteil der letzteren Variante liegt in der möglichen Autovervollständigung in der IDE, wenn diese Fluid in HTML unterstützt.

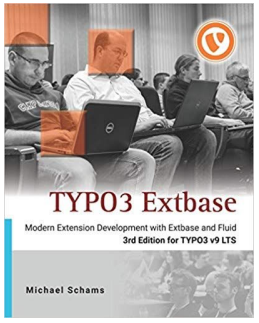
Links und Hilfe

Links

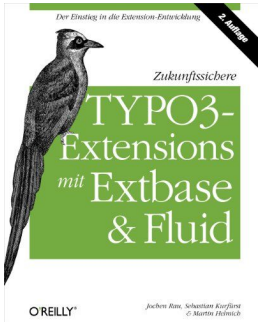
Titel	Links
TYPO3 Coding Guidelines	https://docs.typo3.org/m/typo3/reference-coreapi/master/en-us/CodingGuidelines/Index.html
Extbase und Fluid Cheatsheet	http://www.lobacher.de/files/cs/ExtbaseFluidCheatSheet_3.02_pluswerk.pdf
Fluid Dokumenation	https://docs.typo3.org/m/typo3/guide-extbasefluid/master/en-us/Fluid/Index.html
Fluid Inline-Converter	http://www.fluid-converter.com/
TCA-Reference	https://docs.typo3.org/m/typo3/reference-tca/master/en-us/
Unit Tests in TYPO3	https://docs.typo3.org/m/typo3/reference-coreapi/master/en-us/Testing/WritingUnit.html
DDEV in TYPO3	https://docs.typo3.org/m/typo3/guide-contributionworkflow/master/en-us/Appendix/SettingUpTypo3Ddev.html
Composer in TYPO3	https://docs.typo3.org/m/typo3/guide-installation/master/en-us/QuickInstall/Composer/Index.html und https://composer.typo3.org

Literatur

Wir können vor allem zwei Werke zu diesem Thema empfehlen:



TYPO3 Extbase: Modern Extension Development for TYPO3 CMS with Extbase and Fluid
Sprache: Englisch
ISBN: 1099083079



Zukunftssichere TYPO3-Extensions mit Extbase und Fluid
Sprache: Deutsch
ISBN: 3955614697