FACHHOCHSCHULE FLENSBURG

Fachbereich Angewandte Informatik

BACHELORTHESIS

im Studiengang Medieninformatik

Thema:	Integration der Datenbank-Abstraktionsschicht Doctrine2 in das Content-Management-System TYPO3
eingereicht von:	Stefan Kowalke <stefan.kowalke@stud.fh-flensburg.de>
Matrikelnummer:	485366
Abgabedatum:	4. Mai 2014
Erstprüfer:	Prof. Dr. Hans-Werner Lang
Zweitprüfer:	Dipl. VK Tobias Hiep

Dieses Dokument wurde am 4. Mai 2014 mit LaTeX 2ε gesetzt.

Schrift:	12pt-TODO
Typographie:	2012/07/29 v3.11b KOMA-Script
System:	LuaTeX-0.76.0 auf OSX 10.8
Editor:	Mou 0.8.5 beta und VIM 7.4

ABSTRACT

In der Defaulteinstellung speichert das CMS TYPO3 die Inhalte der Website in einer MySQL-Datenbank. Soll sich das CMS mit der Datenbank eines anderen Herstellers verbinden, muss die mitgelieferte Systemextension *DBAL* installiert werden. Damit die Anfrage von der DBAL Extension geparst und in die anderen SQL-Dialekte konvertiert werden kann, müssen sich die Entwickler - sei es TYPO3 Kern- oder externer Extensionprogrammier - an gewisse Richtlinien, wie zum Beispiel die ausschließliche Nutzung der TYPO3 Datenbank API, halten. Trotz der Nutzung der API können zu DBAL inkompatible Anfragen formuliert werden. Ferner ist es möglich komplett an der API vorbei mit der Datenbank zu kommunizieren, wodurch man wieder beim Anfangsproblem angelangt ist, welches man mit der Nutzung der DBAL Extension lösen wollte: Die Fixierung auf einen SQL-Dialekt. Desweiteren verwendert die DBAL Extension die extene Bibliothek *AdoDB* welche nicht mehr aktiv weiterentwickelt wird und weder Verbessungen noch Sicherheitsupdates bereitstellt. Ziel dieser Arbeit war es zum einen die veraltete Bibliothek gegen eine andere - in dem Fall Doctrine 2 DBAL - zu ersetzen, die in sich unter aktiver Entwicklung befindet, und zum anderen eine einheitlichen Zugriff auf die Datenbank zu erlauben, der die direkte Kommunikation mit der Datenbank verbietet. Es wurde eine Extension geschrieben, die die TYPO3 eigene Datenbank API überschreibt und anstelle dessen eine eigene API anbietet, die über Doctrine DBAL und PDO auf die Datenbank zugreift. Dabei wurde auf die Abwärtskompatibilität zur TYPO3 eigenen API geachtet, dass noch nicht angepasster Code weiterhin mit der Datenbank kommunizieren kann. Zu Test- und Demonstrationszwecken wurde der Kern von TYPO3 an die neue API Dies erlaubt das manuelle Testen der Funktionsweise des Backends und des Frontends - es wurden jedoch auch Unit Tests für die API implementiert, wodurch das auch das automatische Testen ermöglicht wird.

Version: Commit: 5056300 from 2014-05-03 20:43:55 +0200

"War is peace.
Freedom is slavery.
Ignorance is strength."

– George Orwell, 1984

1

EINLEITUNG

Als sich auf den Developer Days 2006 das Entwicklerteam für einen Nachfolger der eben erst erschienen TYPO3 Version 4.0 formierte
(vgl. [TYP08]), war wohl keinem der dort Anwesenden klar wohin die Reise gehen würde – ging man anfänglich noch von einem Refactoring[1] der schon vorhandenen Codebasis aus.

In der Konzeptionsphase kristallisierte sich immer mehr heraus, dass es damit nicht getan sein würde. Der Nachfolger mit dem Arbeitstitel "Phoenix" sollte nicht nur den zukünftigen Anforderungen des Web standhalten, sondern die Position der Version 4.0 weiter ausbauen. Das Entwicklerteam um Chefentwickler Robert Lemke entschloss sich die Version 5.0 des Systems komplett neu zu schreiben [Quelle anfügen] und merkte dabei, dass Entwickler bei der Programmierung von Webanwendungen immer wieder mit den gleichen Problemen wie Routing, die Erstellung und Validierung von Formularen, Login von Benutzern oder dem Aufbau einer Verbindung zur Datenbank konfrontiert werden.

Die Idee eines – von dem Content-Management-System – unabhängigen PHP Frameworks war geboren und wurde zunächst auf den Namen FLOW3 getauft. Dieses Framework sollte die spätere Basis für TYPO3 5.0 bilden und all die oben beispielhaft angeführten wiederkehrenden Aufgaben übernehmen. Die Version 5.0 von TYPO3 sollte lediglich eins von vielen Packages darstellen mit denen FLOW3 erweitert werden kann. Vielmehr wurde es als eigenständiges "Webapplication Framework" konzipiert und umgesetzt, so dass es auch ohne ein Content-Management-Sytem (CMS) betrieben werden kann und auch wird. [Quelle zu Rossmann einfügen].

Schon in einer recht frühen Entwicklungsphase hat man sich dem Thema Persistenz gewidmet, die zunächst noch als "Content Repository for Java Technology API (JCR)" in PHP implementiert, jedoch später wegen zu vieler Probleme bei der Portierung der Java Spezifikation JSR-170 nach PHP durch eine eigene Persistenzschicht ersetzt wurde (vgl. [Dam10]). Im weiteren Verlauf der Entwicklung kam man von dieser Idee wieder ab, da die eigene Persistenzschicht nicht performant genug war und andere Projekte wie Doctrine oder Propel schon fertige Lösungen anboten (vgl. [DEM14]). Schlußendlich entschied man sich für die Integration von Doctrine als Persistenzschicht, da der Hauptentwickler von Doctrine, Benjamin Eberlei, seine Hilfe anbot.

Für die Anwender stellt sich bei einem Versionssprung stets die Frage, ob eine Migration von der alten zur neuen Version möglich ist und mit wieviel Aufwand dies verbunden sein würde. Diesen Bedenken folgend trafen sich die Kernentwickler beider Teams 2008 in Berlin, um die Routemaps beider Projekte in Einklang zu bringen. Als ein Ergebnis dieses Treffens wurde das "Berlin Manifesto"(vgl. [TYP08]) bekanntgegeben, welches mit kappen Worten feststellt[2]:

1 Strukturverbesserung des Quellcodes bei Beibehaltung der Funktionalität

2 Mittlerweile wird TYPO3 Neos innerhalb der Community nicht mehr als der Nachfolger von TYPO3 CMS angesehen. Es stellt lediglich – wie TYPO3 Flow – ein weiteres Produkt innerhalb der TYPO3 Familie dar. [Quellen angabe]

> - TYPO3 v4 continues to be actively developed
> - v4 development will continue after the the release of v5
> - Future releases of v4 will see its features converge with those in TYPO3 v5
> - TYPO3 v5 will be the successor to TYPO3 v4
> - Migration of content from TYPO3 v4 to TYPO3 v5 will be easily possible
> - TYPO3 v5 will introduce many new concepts and ideas. Learning never stops and we'll help with adequate resources to ensure a smooth transition
>
> *Die TYPO3 Kernentwickler*

An der Umsetzung wurde sofort nach dem Treffen begonnen, indem Teile des FLOW3 Frameworks nach TYPO3 Version 4.0 zurück portiert und unter dem Namen *Extbase* als Extension veröffentlicht wurden. Es erfüllt zu gleichen Teilen die Punkte 3 und 6 des Manifests, da es die neuen Konzepte aus FLOW3 der Version 4.0 zur Verfügung stellt und somit gleichzeitig diese Version näher an die Technologie des Frameworks heranführt.

Die Aufgabe von Extbase besteht darin ein Application Programming Interface (API) bereitzustellen, mit denen Entwickler von Extensions auf die internen Ressourcen und Funktionen von TYPO3 CMS zugreifen und das System somit nach eigenen Wünschen und Anforderungen erweitern können, ohne den Code des CMS selbst verändern zu müssen. Es ist als vollständiger Ersatz der bis dahin angebotenen PI-Base API [LINK ZU PI BASE] konzipiert worden, wobei es aktuell noch möglich ist sich für einen der beiden Ansätze zu entscheiden.

Extbase führt per Definition einige – bis dahin in TYPO3 v4 unbekannte – Programmierparadigmen ein. Als größter Unterschied zu dem PI-Based Ansatz ist hier sicherlich das Model-View-Controller (MVC) Pattern zu nennen. Dabei werden die Daten im Model vorgehalten, der View gibt die Daten aus und der Controller steuert die Ausgabe der Daten. Das Model ist unabhängig von der View, was bedeutet, dass die gleichen Daten auf verschiedene Weise ausgegeben werden können. Man denke hier an Meßdaten, die zum einen als Tabelle über einer Listview dargestellt werden können oder als Diagramme mit einer entsprechenden View.

Das Model – eine herkömmliche PHP Klasse – wird dabei von Extbase automatisch auf die Datenbank abgebildet, so dass ein Objekt eine Zeile darstellt und dessen Eigenschaften als Spalten der Tabelle interpretiert werden. Diese Technik wird als Objektrelationale Abbildung (engl. Object-relational mapping (ORM)) genannt. Das zum Einsatz kommende ORM ist Bestandteil der oben erwähnten selbstgeschriebenen Persistenzschicht von FLOW3, da Extbase zu der Zeit rückportiert wurde, als diese bei FLOW3 im Einsatz war.

Obwohl Extbase beständig weiterentwickelt wird und es der Wunsch der Community ist, die in darin verwendete Persistenzschicht gegen Doctrine 2 auszutauschen, was sich in Form von Posts auf der Mailingliste (vgl. [TYP13]) oder in Prototypen ausdrückt (vgl. [Mar12] und [Ebe12]), ist dies bis heute noch nicht realisiert worden. Der Chefentwickler von Doctrine, Benjamin Eberlei, hat gegenüber dem Autor in einer persönlichen

Korrespondenz die unterschiedlichen Ansätze beider Projekte wie folgt zum Ausdruck gebracht:

> (…) Doctrine nutzt das Collection interface, Extbase SplObjectStorage.Doctrine Associationen funktionieren semantisch anders als in Extbase, z.B. Inverse/Owning Side Requirements. Typo3 hat die Enabled/Deleted flags an m_n tabellen, sowie das start_date Konzept. Das gibts in Doctrine ORM alles evtl nur über Filter API, aber vermutlich nicht vollständig abbildbar. Das betrifft aber alles nur das ORM, das Doctrine Database Abstraction Layer (DBAL) hinter Extbase zu setzen ist ein ganz anderes Abstraktionslevel.
>
> *Benjamin Eberlei per E-Mail vom 17.12.13 00:12*

Zum jetzigen Zeitpunkt wird die DBAL in TYPO3 durch eine Systemextension [Glossareintrag] bereitstellt, die auf der externen Bibliothek AdoDB basiert, welche jedoch Anzeichen des Stillstands aufzeigt und davon ausgegangen werden kann, dass das Projekt nicht weiterentwickelt wird. [Linkt zu SourceForge]

Anhand dieser Fakten wird ersichtlich, dass die Integration von Doctrine erstrebenswert ist, da dadurch die Abhängigkeit zu dem inaktiven Projekt AdoDB aufgelöst werden kann. Da jedoch eine Integration von Doctrine ORM in Extbase nicht in der gegebenen Zeit, die für die Bearbeitung der Thesis zur Verfügung steht, zu realisieren ist, wurde der Fokus stattdessen auf die Integration von Doctrine CMS in TYPO3 gelegt, wodurch nicht nur Extbase von den Möglichkeiten eines DBAL profitieren kann, sondern der gesamte Core und somit alle Extensions die noch nicht mit Extbase erstellt worden sind.

Ferner wird durch diesen Ansatz eine stabile Basis zu Verfügung gestellt, auf der eine zukünftige Integration der ORM Komponente von Doctrine in Extbase aufbauen kann.

Ziel dieser Thesis ist es einen funktionierenden Prototypen zu entwickeln, der zum einen aus einer Extension besteht, die für die Integration von Doctrine DBAL zuständig ist und zum anderen aus einem modifizierten TYPO3, welches die neue API, die mit der Extension eingeführt wird, beispielhaft benutzt.

Im ersten Teil werden die eingesetzten Werkzeuge vorgestellt. Es wird erklärt warum diese und nicht andere eingesetzt worden sind und wie diese in Hinblick auf die Aufgabenstellung benutzt wurden.

Der zweite Teil beschreibt die praktische Umsetzung und schließt mit einer Demonstration wie der Prototyp getestet werden kann.

Teil drei gibt einen Ausblick auf die weitere Verwendung des Quellcodes und des Prototypen, während Teil vier mit einem Fazit schließt.

Im Anhang befindet sich neben den obligatorischen Verzeichnissen für Literatur und Abbildungen ein Glossar, sowie das Abkürzungsverzeichnis.

2

GRUNDLAGEN

2.1 *TYPO3 CMS*

2.1.1 *Geschichte*

TYPO3 CMS ist ein Web Content Management-Sytem (WCMS) und wurde von dem dänischen Programmierer Kaspar Skårhøj im Jahr 1997 zunächst für seine Kunden entwickelt - im Jahr 2000 von ihm unter der GNU General Public License v.2 (GPL2) veröffentlicht. Dadurch fand es weltweit Beachtung und erreichte eine breite Öffentlichkeit. Laut der Website T3Census[1] gab es am 7. April 2014 208561 Installationen von TYPO3 CMS.

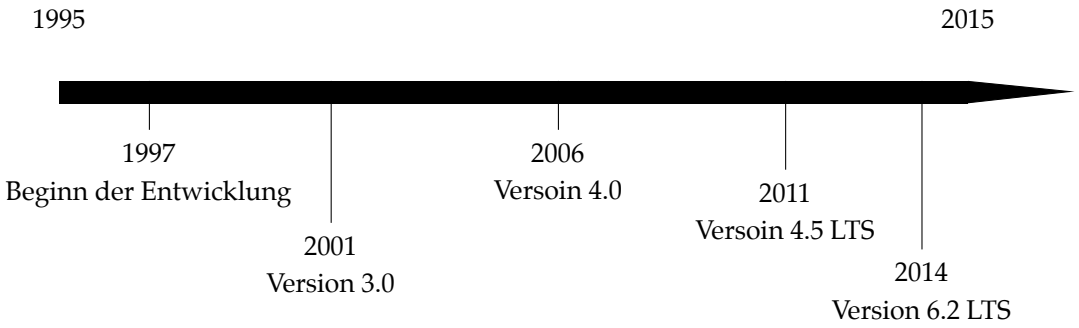

Abbildung 2.1: Zeitachse der TYPO3 CMS Entwicklung

Im Jahr 2012 entschied sich das Projekt zu einer Änderung in der Namesgebung:

- aus TYPO3 v4[2] wurde TYPO3 CMS
- aus FLOW3 wurde TYPO3 Flow
- und aus TYPO3 5.0 / TYPO3 Phoenix wurde TYPO3 Neos

Diese Änderung wurde notwendig, da schon länger abzusehen war, dass TYPO3 Phoenix nicht den Nachfolger von TYPO3 v4 darstellt. Somit war die Entwicklung von TYPO3 v4 in dem Versionszweig 4.x gefangen und es konnten keine neuen Features eingebaut oder veraltete Funktionen entfernt werden. Durch dieses neue Schema bekommt der Name "TYPO3" die Bedeutung einer Dachmarke zuteil, während "TYPO3 CMS", "TYPO3 Flow" und "TYPO3 Neos" Produkte innerhalb der TYPO3 Familie darstellen. Im weiteren Verlauf dieser Arbeit werden ausschließlich die neuen Namen verwendet.

1 http://t3census.info/
2 Damit ist das von Skårhøj entwickelte CMS gemeint, welches den 4.x Zweig des Projekts darstellt.

Heute kümmert sich ein Team um die Entwicklung von TYPO3 CMS und eines um TYPO3 Flow und TYPO3 Neos. Dahinter steht keine Firma, wie es bei anderen Open Source Projekten wie Drupal (Acquia) oder Wordpress (Automattic) vorzufinden ist, sondern die TYPO3 Association (T3Assoc). Die T3Assoc ist ein gemeinnütziger Verein und wurde 2004 von Kaspar Skårhøj und anderen Entwicklern gegründet um als Anlaufstelle für Spenden zu dienen, die die langfristige Entwicklung von TYPO3 sicherstellen sollen. Die Spenden werden in Form von Mitgliedsbeiträgen erhoben.[3]

2.1.2 *Definition*

TYPO3 CMS ist ein klassisches CMS, welches auf die Erstellung, die Bearbeitung und das Publizieren von Inhalten im Intra- oder Internet spezialisiert ist und es somit per Definition zu einem WCMS macht.

Daneben findet man auch die Bezeichnung Enterprise Content Management-Sytem (ECMS)[4], was als Hinweis auf den Einsatz des Systems für mittel- bis große Webprojekte dient.

Als letztes sei noch erwähnt, dass TYPO3 CMS ebenso zu den Content Management Framework (CMF) gezählt werden kann, da es dem Entwickler verschiedene APIs zur Verfügung stellt. Dieser Begriff findet sich unter anderen in einem Kommentar im – von TYPO3 CMS erzeugten – HTML-Code:

```
<!--
    This website is powered by TYPO3 - inspiring people to share!
    TYPO3 is a free open source Content Management Framework
    initially created by Kasper Skaarhoj and licensed under GNU/GPL.
    TYPO3 is copyright 1998-2012 of Kasper Skaarhoj. Extensions
    are copyright of their respective owners.
    Information and contribution at http://typo3.org/
-->
```

2.1.3 *Architektur und Aufbau von TYPO3 CMS*

Im folgenden werden die grundlegenden Konzepte von TYPO3 CMS vorgestellt. Dort wo es für das weitere Verständnis notwendig ist, wird tiefer in das Thema eingestiegen. Ansonsten werden die Konzepte lediglich angerissen um einen generellen Überblick zu erhalten.

Webstack als Basis

TYPO3 CMS wurde in PHP: Hypertext Processor (PHP) - basierend auf dem Konzept der Objektorientierung - geschrieben und ist damit auf jeder Plattform lauffähig, die über einem PHP Interpreter verfügt. Die Version 6.2 von TYPO3 CMS benötigt mindestens PHP 5.3.7.

3 http://association.typo3.org/
4 http://www.typo3.org

PHP bildet zusammen mit einem Apache Webserver und einer MySQL Datenbank den sogenannten Webstack, der abhängig von dem eingesetzten Betriebssystem MAMP (OSX / **M**ac), LAMP (**L**inux) oder WAMP (**W**indows) heißt.

In der Standardeinstellung kommt MySQL als Datenbank zum Einsatz - durch die Systemextension[5] *DBAL* können jedoch auch Datenbanken anderer Hersteller angesprochen werden. Eine genaue Analyse dieser Extension erfolgt im Kapitel **??**[Kapitel zur Analyse von ext:DBAL einfügen].

Ansichtssache

Aus Anwendersicht teilt sich TYPO3 CMS in zwei Bereiche:

- das Backend
  stellt die Administrationsoberfläche dar. Hier erstellen und verändern Redaktuere die Inhalte; während Administratoren das System von hier aus konfigurieren

- das Frontend
  stellt die Website dar, die ein Besucher zu Gesicht bekommt.

(vgl. [DRB08, S. 5])

Der Systemkern und die APIs

TYPO3 CMS besteht aus einem Systemkern, der lediglich grundlegende Funktionen zur Datenbank-, Datei- und Benutzerverwaltung zu Verfügung stellt. Dieser Kern ist nicht monolithisch aufgebaut, sondern besteht aus Systemextensions. (vgl. [Lab+06, S. 32])

5 Eine kurze Einführung in die verschiedenen Arten von Extension findet sich im Kapitel 2.1.4

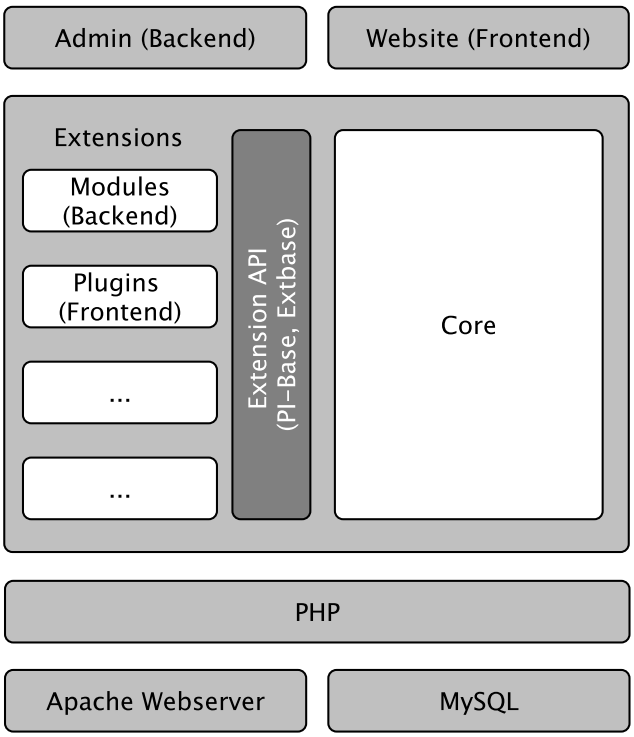

Abbildung 2.2: Schematischer Aufbau von TYPO3

Die Gesamtheit aller von TYPO3 CMS zur Verfügung gestellten APIs, wird als die *TYPO3 API* bezeichnet. Diese kann - analog zum Konzept von Backend und Frontend - in eine *Backend API* und eine *Frontend API* unterteilt werden kann. Die Aufgabe der Frontend API ist die Zusammenführung der getrennt vorliegenden Bestandteile (Inhalt, Struktur und Layout) aus der Datenbank oder dem Cache zu einer HTML-Seite. Die Backend API stellt Funktionen zur Erstellung und Bearbeitung von Inhalten zur Verfügung. (vgl. [DRB08, S. 5 ff.])

Die APIs, die keiner der beiden Kategorien zugeordnet werden kann, bezeichnet Dulepov [DRB08, S. 5 ff.] als *Common-API*. Die Funktionen der Common-API werden von allen anderen APIs genutzt. Ein Beispiel dafür stellt die Datenbank API dar, welche in der Regel nur einfache Funktionen wie das Erstellen, Einfügen, Aktualisieren, Löschen und Leeren[6] von Datensätzen bereitzustellen hat. Würde man je eine Datenbank API für das Frontend und das Backend zur Verfügung stellen, bricht man eine wichtige Regel der Objekt-orientierten Programmierung - Don't repeat yourself. Dieser - mit hoher Wahrscheinlichkeit - redundante Code würde die Wartbarkeit des Programms verschlechtern und die Fehleranfälligkeit erhöhen.

Auf die aktuelle Datenbank-API wird in Kapitel **??**[KAPITEL zur Analyse der aktuellen Situation einfügen] näher eingegangen.

Verzeichnisstruktur

Im Gegensatz zu früheren TYPO3 CMS Versionen gibt es kein *Dummy-Package*[7] mehr. Ab Version 6.2 enthält der Download lediglich den TYPO3 CMS Kern in Form des Verzeichnisses `typo3/`.

6 CRUD - **C**reate, **R**etrieve, **U**pdate und **D**elete

7 Damit ist ein weitgehend leeres Paket gemeint, dass alle Dateien enthält die im Webroot des Servers laufen sollen. Es stellt einen Container für die spätere Website dar.

Dieses Verzeichnis ist außerhalb des Webroots abzulegen. Im Webroot ist ein Verzeichnis `www.example.com` anzulegen, in dem die Verzeichnisse `fileadmin/`, `typo3conf/`, `typo3temp/` und `uploads/` anzulegen sind. Das Verzeichnis `typo3_src/` ist ein (Linux-)Symlink auf das Installationsverzeichnis von TYPO3 CMS und das Verzichnis `typo3/` ist ebenfalls ein Symlink, welcher über den Symlink `typo3_src` auf `typo3` zeigt. Dieser Aufbau macht ein Update recht einfach, da lediglich der Symlink `typo3_src` auf das Verzeichnis der neuen Version rekonfiguriert werden muss.

```
.
├── Packages/
│   └── Libraries/
├── fileadmin/
├── typo3_src/ -> ../../typo3-6.2.0
├── typo3/ -> typo3_src/typo3
│   ├── contrib/
│   ├── ext/
│   ├── gfx/
│   ├── install/
│   ├── js/
│   ├── mod/
│   └── sysext/
├── typo3conf/
│   ├── ext/
│   │   ├── doctrine_dbal/
│   │   └── phpunit/
│   └── l10n/
├── typo3temp/
├── uploads/
├── media/
├── pics/
├── tf/
└── tx_phpunit/
```

Abbildung 2.3: Verzeichnisstruktur von TYPO3 CMS

Im folgenden werden die einzelnen Verzeichnisse näher erklärt:

Im Verzeichnis `www.example.com` muss noch ein Symlink `index.php` angelegt werden, welcher auf `typo3_src/index.php` zeigt.

Unter `www.example.com/typo3conf/` befindet sich die Datei `LocalConfiguration.php`. Diese enthält die Grundkonfiguration in Form eines Arrays. Darin sind verschiedenen Einstellungen festgelegt:

- Debug Mode
- Sicherheitslevel für den Login (Fronend und Backend)
- das Passwort für das Installtool (mit MD5 und Salt gehasht)
- die Zugangsdaten zur Datenbank (Benutzername, Password, Datenbankname, Socket, …)
- Einstellungen zum Caching
- Titel der Website

Verzeichnis	Erklärung
Packages/Libraries/	Dieser Ordner wurde von der von TYPO3 Flow übernommen. In einer der nächsten Versionen sollen hier die Extensions gespeichert werden. Im aktuellen Fall liegen hier externe Bibliotheken, die mit Composer[8] installiert wurden, wie zum Beispiel Doctrine
fileadmin/	In diesem Ordner werden Dateien gespeichert, die über die Website erreichbar und ausgeliefert werden sollen. Dazu zählen CSS-, Image, HTML-Template- und TypoScriptdateien. Allgemein also Dateien, die vom Websitebetreiber hochgeladen werden.
typo3/	Der TYPO3 CMS Kern
contrib/	Bibliotheken von Drittanbietern
contrib/ext/	Das Verzeichnis für globale Extensions
gfx/	Jegliche Grafiken, die im Core verwendet werden
gfx/install/	Hier befand sich in früheren Versionen das Installtool. Aktuell existiert das Verzeichnis nur noch aus Kompatibilitätsgründen und wird in einer der nächsten Versionen entfernt. Das Installtool wurde als Sytemextension realisiert und ist im entsprechenden Ordner unter `sysext/install/` zu finden
gfx/js/	Hier befinden sich die JavaScript Bibliotheken, die von Core genutzt werden.
gfx/mod/	Enthält die Konfiguration der Hauptmodule des Backends (File, Help, System, Tools, User, Web).
sysext/	Enthält die Systemextensions.
typo3conf/	Lokale Extensions und die lokale Konfiguration
typo3temp/	Temporäre Dateien
uploads/	Dateien die vom Websitebesucher hochgeladen werden - zum Beispiel über ein Formular.

Tabelle 2.1: Erläuterung der Verzeichnisstruktur von TYPO3 CMS

- Einstellungen zum Erzeugen von Graphiken

Die Einstellungen zur Datenbank werden im praktischen Teil näher beleuchtet.

TCA

Wie bereits geschrieben wurde, stellt das Backend (BE) eine Ansicht auf die Datenbank dar. Die Inhalte werden dabei mittels Formulare eingegeben und in der Datenbank gespeichert. Die Konfiguration dieser Formulare erfolgt über ein globales PHP-Array - dem Table Content Array (TCA).

Über das TCA werden die Metadaten einer Tabelle (Datentyp, Länge, Engine) mit weiteren Daten angereichert. So können mit dem TCA

- die Beziehungen einer Tabelle zu anderen Tabellen beschrieben werden
- in welchem Layout soll ein Feld im Formular dargestellt werden
- und wie soll das Feld validiert werden.

Enthält eine Tabelle keinen Eintrag im TCA ist sie im Backend nicht sichtbar.(vgl. [TYP])

[HINWEIS: Vielleicht kann das TCA auch einfach im Glossar beschrieben werden. Einen Eintrag gibt es schon unter tcag]

XCLASS

TYPO3 CMS besitzt einen Mechanismus, der es erlaubt Klassen zu erweitern oder Methoden mit eigenem Code zu überschreiben. Dies funktioniert für den Systemkern wie auch für andere Extensions. Dieses Feature nennt sich XCLASS und wird vom Prototypen eingesetzt um die Datenbankklasse von TYPO3 CMS zu überschreiben. Darauf wird im Kapitel [KAPITEL Analyse Ist-Zustand einfügen] näher eingegangen. Hier soll lediglich der Hintergrund zu XCLASS beschrieben werden.

Damit eine Klasse per XCLASS erweiterbar ist, darf sie nicht per `new()` Operator erzeugt werden, sondern mit der von TYPO3 CMS angebotenen Methode `\TYPO3\CMS\Core\Utility\GeneralUtility::makeInstance()`. Diese Methode sucht im globalen PHP-Array `$GLOBALS['TYPO3_CONF_VARS']['SYS']['Objects']` nach angemeldeten Klassen, instanziiert diese und liefert sie anstelle der Originalklasse zurück. Dieses Array dient der Verwaltung der zu überschreibenden Klassen und erfolgt in der Datei `ext_localconf.php` innerhalb des Extensionsverzeichnisses[9].

Der Mechanismus hat jedoch ein paar Einschränkungen:

- der Code der Originalklasse kann sich ändern. Es ist somit nicht sichergestellt, dass der überschreibende Code weiterhin das macht, wofür gedacht war
- XCLASSes funktioneren nicht mit statischen Klassen, statischen Methoden und finalen Klassen
- eine Originalklasse kann nur einmal per XCLASS übeschrieben werden
- einige Klassen werden sehr früh bei der Initialisierung des System instanziiert. Das kann dazu führen, dass Klassen die als Singleton ausgeführt sind, nicht überschrieben werden können oder es kann zu unvorhergesehenen Nebeneffekten kommen.

9 Zur Erläuterung des Aufbaues einer Extension siehe Kapitel 2.2.

2.1.4 *Extensions*

Extensions sind funktionale Erweiterungen. Sie interagieren mit dem Systemkern über die Extension API und stellen die Möglichkeit dar TYPO3 CMS zu erweitern und anzupassen.

Extensions werden - je nach Kontext - in unterschiedliche Kategorien eingeteilt, die hier kurz vorgestellt werden.

Einteilung

Systemextension werden mit dem System mitgeliefert und befinden sich ausschließlich im Ordner `typo3/sysext/`. Sie werden nochmals unterteilt in jene, die für den Betrieb von TYPO3 CMS unabdingbar sind und solche die nicht zwangsläufig installiert sein müssen, jedoch wichtige Funktionen beisteuern. Die Extension DBAL ist in die letzte Kategorie einzuordnen. Auf sie wird im Kapitel **??** näher eingegangen.

Neben Systemextensions gibt es noch globale[10] und lokale Extensions. Lokale Extensions werden im Ordner `typo3conf/ext/` und globale Extensions im Ordner `typo3/ext` installiert.

Eine weitere Kategorisierung erfolgt nach dem Aufgabengebiet einer Extension. Die Festlegung auf eine der folgenden Kategorien hat keine direkte Auswirkung auf die Funktion der Extension. Sie wird von TYPO3 CMS hauptsächlich als Sortiermerkmal im Extension Manager (EM) genutzt.

[Ich denken diese Übersicht kann gelöscht werden, da irrelevant]

- Frontend
- Frontend Plugins
- Backend
- Backend Modul
- Service
- Example
- Templates
- Documentation
- Verschiedenens

10 Da globale Extensions nur in bestimmten Szenarien einen Sinn ergeben und in der Realität so gut wie nicht vorkommen, wird von der TYPO3 Community der Begriff "Extension" synonym zum Begriff "lokale Extension" verwendet. Die Arbeit folgt dieser Regelung.

```
.
├── Classes/
│   ├── Install/
│   ├── Loggers/
│   └── Persistence/
├── Configuration/
│   ├── ExtensionBuilder/
│   ├── TCA/
│   └── TypoScript/
├── Documentation/
├── Resources/
│   ├── Private/
│   └── Public/
├── Tests/
│   ├── Build/
│   └── Unit/
├── vendor/
│   ├── bin/
│   ├── composer/
│   ├── doctrine/
│   └── symfony/
├── composer.json
├── ext_emconf.php
├── ext_icon.gif
├── ext_localconf.php
├── ext_tables.php
├── ext_tables.sql
└── ext_tables_static+adt.sql
```

Abbildung 2.4: Verzeichnisstruktur einer Extension

Extension Manager

Der EM ist ein BE Modul, über das die Extensions verwaltet werden können. Es erlaubt die Aktivierung, Deaktivierung, Herunterladen und das Löschen von Extensions. Darüberhinaus bietet der EM Möglichkeiten zur detailierten Anzeige von Informationen über die Extensions wie das Changelog[11], Angaben zu den Autoren und Ansicht der Dateien der Extension.

Verzeichnisstruktur

Unabhängig von der Einteilung der Extensions in die veschiedenen Kategorien unterscheiden sie sich nicht in der Verzeichnis- und Dateistruktur. Mit der Integration von Extbase in TYPO3 CMS hat sich eine neue Verzeichnisstruktur etabliert. Sie folgt dem Paradigma *Konvention statt Konfiguration*, was bedeutet, dass durch Einhaltung der Struktur keine weitere Konfiguration notwendig ist.

[Beschreibung der Verzeichnisstruktur einfügen]

11 Das Protokoll der Codeänderungen, die ein Programm im Laufe seines Lebens erlebt

Verzeichnis / Datei	Erklärung
Classes/	Hier erwartet TYPO3 CMS alle Klassendateien. Diese können in weiteren Unterverzeichnissen nach ihren Zweck unterteilt werden (Controller, Service, Loggers, Persistence)
Configuration/	Enthält Konfigurationsdateien wie TypoScript oder TCA.
Documentation/	Enthält die Dokumentation im ReST[12] Format aus der HTML und ein PDF generiert werden kann.
Resources/Private/	Enthält die Fluid-Template[13], Sprach- und Sassdateien. Kurz alle Dateien, die für die Struktur und das Aussehen einer Website notwendig sind, jedoch lediglich Templates darstellen und noch verarbeitet werden müssen
Resources/Public/	Hier liegen Grafiken, CSS- und Javasciptdateien
Tests/	Hier werden die PHPUnit-, Akzeptanz- und/oder Verhaltenstests abgelegt.
vendor/	Dieses Verzeichnis wird von Composer angelegt und enthält externe Abhängigkeiten wie in dem Fall Doctrine DBAL. Im Moment ist dies redunant, da der selbe Inhalt auch im Webroot der Site unter `Packages/Libraries/` verfügbar sein muss. Das liegt an der noch nicht vollständig umgesetzten Kompatibilität von TYPO3 CMS zu Composer, die jedoch in einer späteren Version noch nachgereicht wird.
composer.json	Seit Version 6.2 von TYPO3 CMS ist eine Composer.json erforderlich. In ihr werden Metadaten der Extensions wie Name, Typ (System- oder lokale Extension), Lizenz, Version und Abhängigkeiten definiert. Sie wird in Zukunft wahrscheinleich die Datei `ext_emconf.php` ablösen.
ext_emconf.php	Diese Datei ist unabdinglich für die Funktionsweise der Extensions. Sie definiert ebenso wie die `composer.json` Metadaten jedoch in einem PHP-Array und nicht im JSON-Format[14]. Diese Datei existiert seit den Anfängen von TYPO3 CMS und wird wahrscheinlich bald von der composer.json abgelöst werden.
ext_icon.gif	Ein Icon für die Extension, welches im BE angezeigt wird
ext_localconf.php	In dieser Datei wird die Extension für einen Hook oder eine XCLASS registiert.
ext_tables.php	Die Datei hat drei Aufgaben: • Definition von Extensionstabellen • Definition von Feldern und Tabellen, die von dieser Extension erweitert werden • Frontend (FE) Plugins und BE Module werden hier registiert

12 reStructuredText http://docutils.sourceforge.net/rst.html
13 TYPO3s Templating Sprache
14 http://json.org/

ext_tables.sql	Diese Datei enthält Anweisungen um eine Datenbanktabelle zu erstellen. Sie sind im MySQL Format zu formulieren, auch wenn als Datenbank etwas anderes genutzt wird. TYPO3 CMS parst die Datei mit einem eigenen (sehr rudimentären) SQL-Parser und generiert einen eigene SQL-Abfrage. Dieses Vorgehen hat den Hintergrund, dass dadurch auf Fehler im diesen Dateien reagiert werden kann und zum anderen das die mitgelieferte Datenbankabstraktionsschicht die Anweisungen in das SQL der entsprechenden Hersteller übersetzen kann. Auf dem Inhalt dieser Datei wird im praktischen Teil in Kaptitel **??** [KAPITEL REFERENZ einfügen] noch genauer eingegangen.
ext_tables_static+adt.sql	Auch diese Datei enthält SQL Code wie `ext_tables.sql`. Der Unterschied besteht darin, dass sie `INSERT` Statements enhalten kann um statische Daten in eine Tabelle bei der Installation einer Extensions einzufügen. Als Beispiel sei hier der EM genannt, in dessen Tabelle wird über diese Datei die URL zum TYPO3 Extension Repository (TER) eingefügt. Auf dem Inhalt dieser Datei wird im praktischen Teil in Kaptitel **??** [KAPITEL REFERENZ einfügen] noch genauer eingegangen.

Tabelle 2.2: Erläuterung der Verzeichnisstruktur einer Extension

2.2 *Doctrine*

Das Doctrine Projekt besteht aus einer Reihe von PHP-Bibliotheken, die Schnittstellen rund um die Datenbankschicht bereitstellen. Die Konzepte sind beeinflusst von Javas *Hibernate*[15] (vgl. [T3N09]) und dem Entwurfsmuster *Active Record*, welches von Martin Fowler [Fow03] vorgestellt wird.

Das Projekt wurde 2006 von Konsta Vesterinen initiiert[16] und im Jahr 2008 als Version 1.0.0 veröffentlicht. Die Version 2.0 wurde im Jahr 2010 unter dem neuen Projektleiter Benjamin Eberlei fertiggestellt.

Die beiden bekanntesten Produkte des Projekts sind:

- ORM - ermöglicht die objekrelationale Abbildung von Objekten auf Datenbanktabellen
- DBAL - stellt eine Datenbankabstraktionsschicht bereit

Wie aus Abbildung 2.5 ersichtlich wird, ist Doctrine DBAL lediglich als eine dünne Schicht auf Basis der PHP-Extension PHP Data Objects (PDO) ausgeführt, die die grundlegenden Funktionen zur Abstraktion von Datenbanken implementiert. Die ORM Schicht baut auf Doctrine DBAL auf.

15 http://hibernate.org/
16 http://docs.doctrine-project.org/projects/doctrine1/en/latest/en/manual/acknowledgements.html

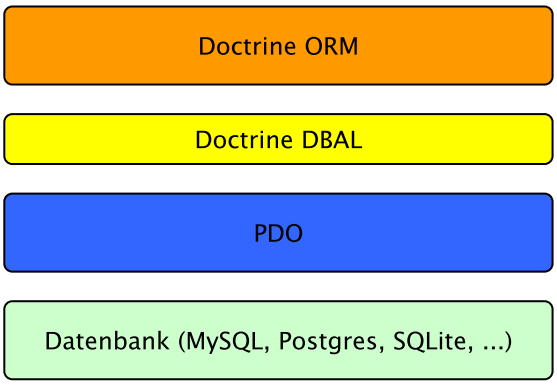

Abbildung 2.5: Schematischer Aufbau von Doctrine

2.2.1 *ORM*

> Ein ORM ist eine Abstraktionsschicht zwischen relationaler Datenbank und der eigentlichen Anwendung. Statt per SQL kann man durch das ORM objektorientiert auf die Daten zugreifen.
>
> *Jonathan Wage [T3N09]*

Das folgende Anwendungsbeispiel zeigt eine typische Situation. Es wird zunächst ein neues Objekt eines Studenten erzeugt um im weiteren Verlauf mit verschiedenen Daten angreichert. Anschließend wird es als zu speicherndes Objekt bei der Datenbank registiert und schlußendlich gespeichert.

Im zweiten Codelisting wird die gleiche Aufgabe auf dem herkömmlichen Weg gelöst. Dabei wird die Anfrage an eine MySQL und PostgreSQL Datenbank gesendet. Die Variable `$connection` enthält je eine initialisierte Verbindung zur entsprechenden Datenbank.

```php
<?php

$student = new Student();
$student->setFirstName('Stefano');
$student->setLastName('Kowalke');
$student->setEnrolmentNumber('12345');
$entityManager->persist($student);
$entityManager->flush();
```

Listing 1: Speichern eines Studenten in die Datenbank mit ORM

Der Code in Listing 1 gibt keinen Rückschluss auf die darunterliegende Datenbank. Die Daten des Studenten könnten in eine CSV-Datei, einer MySQL oder Postgres Datenbank gespeichert worden sein. Hingegen wurden in Listing 2 zwei verschiedene Methoden genutzt, um die Daten in eine MySQL und Postgres Datenbank zu schreiben. Die Speicherung in eine Textdatei wurde dabei nicht berücksichtigt.

Doctrine ORM ist für die Umwandlung des `$student`-Objekt in eine Structured Query Language (SQL)-Abfrage zuständig. Die erzeugte Abfrage ist mit der aus Codebei-

```php
<?php

$sql =
    'INSERT INTO students ('first_name', 'last_name', 'enrolment_number')
    VALUES ('Stefano', 'Kowalke', '12345');

// MySQLi
$result = mysqli_query($connection, $sql);

// PostgreSQL
$result = pg_query($connection, $sql);
```

Listing 2: Speichern eines Studenten in die Datenbank ohne ORM

spiel 2 vergleichbar. Die Konvertierung der Anfrage in die verschiedenen SQL-Dialekte[17] erfolgt durch Doctrine DBAL.

2.2.2 *DBAL*

Doctrine konvertiert das Schema anhand von sehr unterschiedlichen Merkmalen in das SQL der jeweilgen Database Management System (DBMS), die zum Verständnis einen etwas tieferen Einstieg in die Eigenheiten der DBMS erfordern. Da dies den Umfang der Arbeit überschreitet, wurden markante Beispiele gewählt, die den Sachverhalt verdeutlichen.

Datenbankschemas werden in Doctrine von der Klasse `Schema` repräsentiert. Im Beispiel wird zunächst eine Instanz dieser Klasse erstellt und anschließend wird eine neue Tabelle und mehrere Tabellenspalten mit unterschiedlichen Datentypen angelegt. In Zeile 24 wird das Schema in eine SQL-Abfrage übersetzt. Davor existiert es lediglich als PHP-Objekt bis zum Ende der Laufzeit des Scripts.

Die Variable `$myPlatform` enthält die Information über das aktuelle benutzte DBMS.

17 Als SQL Dialekt wird ein vom SQL-Standard abweichender Hersteller-spezifischer Sprachumfang bezeichnet. Ein Dialekt ist in der Regel kompatibel mit dem Standard und erweitert ihn um eigene Sprachkonstrukte.

```php
<?php

$schema = new \Doctrine\DBAL\Schema\Schema();
$beUsers = $schema->createTable('be_users‘);
$beUsers->addColumn('uid', 'integer',
  array('unsigned' => TRUE, 'notnull' => TRUE, 'autoincrement' => TRUE)
);
$beUsers->addColumn('pid', 'integer',
  array('unsigned' => TRUE, 'default' => '0', 'notnull' => TRUE)
);
$beUsers->addColumn('username', 'string',
  array('length' => 50, 'default' => '', 'notnull' => TRUE)
);
$beUsers->addColumn('password', 'string',
  array('length' => 100, 'default' => '', 'notnull' => TRUE)
);
$beUsers->addColumn('admin', 'boolean',
  array('default' => '0', 'notnull' => TRUE)
);
$beUsers->addColumn('history_data', 'text',
  array('length' => 16777215, 'notnull' => FALSE)
);
$beUsers->addColumn('ses_data', 'text',
  array('notnull' => FALSE)
);
$beUsers->setPrimaryKey(array('uid'));
$beUsers->addIndex(array('pid'), 'be_users_pid_idx');
$beUsers->addIndex(array('username'), 'be_users_username');

$queries = $schema->toSql($myPlatform);
```

Listing 3: Erstellen eines Schemas mit Doctrine

Die beiden Listings 4 und 5 zeigen den Inhalt von `$queries` – einmal für MySQL und für PostgreSQL.

```sql
// Der Inhalt von $queries für MySQL
CREATE TABLE `be_users` (
        `uid` int(10) unsigned NOT NULL AUTO_INCREMENT,
        `pid` int(10) unsigned NOT NULL DEFAULT '0',
        `username` varchar(50) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
        `password` varchar(100) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
        `admin` tinyint(1) NOT NULL DEFAULT '0',
        `history_data` mediumtext,
        `ses_data` longtext,
        PRIMARY KEY (`uid`),
        KEY `be_users_pid_idx` (`pid`),
        KEY `be_users_username` (`username`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Listing 4: Das erstellte Schema als MySQL Anfrage

```
// Der Inhalt von $queries für PostgreSQL
CREATE TABLE be_users (
        uid serial NOT NULL,
        pid integer NOT NULL DEFAULT 0,
        username character varying(50) NOT NULL DEFAULT ''::character varying,
        password character varying(100) NOT NULL DEFAULT ''::character varying,
        admin boolean NOT NULL DEFAULT false,
        history_data text,
        ses_data text,
        CONSTRAINT be_users_pkey PRIMARY KEY (uid)
) WITH (
        OIDS=FALSE
);
```

Listing 5: Das erstellte Schema als PostgreSQL

Anhand der Beispiele können folgende Konvertierungen abgeleitet werden:

Abstraktion	MySQL	PostgreSQL
`integer ... auto_increment`	`INT(10) ... AUTO_INCREMENT`	`serial`
`integer`	`INT(10)`	`INTEGER`
`string ... length => 50`	`VARCHAR(50)`	`CHARACTER VARYING(50)`
`boolean`	`TINYINT(1)`	`BOOLEAN`
`text ... length => 255`	`TINYTEXT`	`TEXT`
`text`	`LONGTEXT`	`TEXT`

Tabelle 2.3: Typkonvertierung von Doctrine nach MySQL und PostgreSQL

Doctrine wählt die entsprechenden Typen anhand verschiedener Kriterien aus.

- Angabe von weiteren Optionen: `auto_increment`
- Angabe einer Länge: `length => 50` bzw. `length => 255`
- Konvertierung in äquivalente Datentypen: MySQL implementiert keinen `BOOLEAN` Typ, daher wird hier `TINYINT(1)` genutzt

Dabei ist zu beachten, dass die Werte in Klammern für String-Typen eine andere Bedeutung haben, als für numerische Datentypen. Wird ein `VARCHAR` mit der Länge *34* definiert, bedeutet dies, dass darin eine Zeichenkette mit maximal 34 Zeichen inklusive Leerzeichen gespeichert werden kann. Würde man auf den (vollkommen abwegigen) Gedanken kommen in dieser Spalte das Buch *The Hitchhiker's Guide to the Galaxy* von Douglas Adams speichern zu wollen, würde der Inhalt wie folgt aussehen: "Far out in the uncharted backwate" [Ada95, S. 3]. Der Rest des Textes wird abgeschnitten.

Bei numerischen Datentypen beschreibt der Wert allerdings die Anzeigenbreite - also die Anzahl der angezeigen Ziffern des gespeicherten Wertes. Sollte der in der Spalte gespeicherte Wert kleiner sein als die Länge der Anzeigenbreite, werden die restlichen Stellen nach links mit Leerzeichen aufgefüllt. Wurde die Option `ZEROFILL` gesetzt, werden die restlichen Stellen mit Nullen anstelle von Leerzeichen aufgefüllt. Der Wert beinflusst in keinster Weise den maximal speicherbaren Wert. In einer als `TINYINT` definierten Spalte können immer 256 Werte gespeichert werden. Unabhängig davon ob sie als `TINYINT(1)` oder `TINYINT(4)` deklariert wurde.

Doctrine spiegelt dieses Verhalten wider, indem die Angabe von `length` bei der Deklaration eines Integers dem Wert der Anzeigenbreite entspricht; bei String-Typen den tatsächlichen speicherbaren Wertebreich und bei der Deklaration eines Text-Datentyps als Auswahlkriterium in Bezug auf die unterschiedlichen `*TEXT` Datentypen.

2.2.3 *Unterstützte Datenbank-Plattformen*

Doctrine unterstützt die folgenden Plattformen und kann um weitere Datenbanken erweitert werden.

- DB2
- Drizzle
- MySQL
- Oracle
- PostgreSQL
- SQLAnywhere
- SQLite
- SQLAzure
- SQL Server

2.2.4 *Konzepte und Architektur von Doctrine*

Die beiden Hauptklassen von Doctrine DBAL sind Connection und Statement, die als eine dünne Schicht über PDO implementiert sind. Doctrine nutzt die Funktionen von PDO und erweitert diese um eigene Funktionaliät.

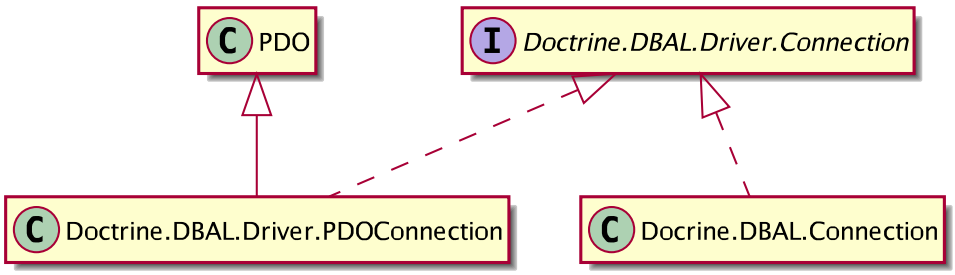

Abbildung 2.6: Schematischer Aufbau von TYPO3

1. [UML Diagramm] √
2. Beschreibung der beiden Klassen
3. Beschreibung von PDO
4. BEschreibung der gemeinsammen Konzepte

- UML Diagramm von Statement und Connection mit Verbindung zu PDO - erweitert PDO -

2.3 *PDO*

Wie wir im Kapitel über Doctrine sehen werden [Hängt davon ab, wo dieses Kapitel eingefügt wird], baut es auf PDO auf. Es verwendet dessen Konzepte und erweitert diese. Um ein tieferes Verständnis von Doctrine DBAL zu erlangen, kommt man nicht umhin sich mit PDO zu beschäftigen.

Im diesem Kapitel wird auf den Hintergrund von PDO. Es werden die Fähigkeiten und Grenzen aufgezeigt, sowie anhand von einfachen Beispielen die Funktionsweise illustriert.

2.3.1 *Was ist PDO*

PHP Data Objects (PDO) stellt eine Abstraktionsbibliothek für die Interaktion mit DBMS verschiedener Hersteller dar und orientiert sich dabei an dem Konzept der Java Data Objects (JDO)[18]. Es ist seit Version 5.1 in PHP enthalten.

Der Grund der Existenz einer solchen Bibliothek liegt darin begründet, dass man in PHP viele verschiedene DBMS über jeweils eigene Extensions ansprechen kann. Die dazu extra zu installierenden Extensions bringt jeweils eine eigene API für die DBMS mit, so das sich die Art des Aufbaues der Verbindung und das Absetzen von Anfragen von DBMS zu DBMS unterschiedlich ausgestaltet ist. Dies kann zu einem hohem Aufwand führen, wenn die Anwendung von einem DBMS auf ein anderes umgestellt werden soll. PDO hingegen definiert unabhängig vom zugrundeliegenden DBMS eine einheitliche Schnittstelle für das Verbindungsmanagement und die Kommunikation mit der Datenbank.

Neben PDO existieren unter anderen die Projekte MDB2[19] und ADOdb[20], die den gleichen Ansatz verfolgen. Wenn man sich jedoch die letzten Releasedaten[21] beider Projekte ansieht, kann man leicht den Eindruck erhalten, dass sie nicht mehr weiterentwickelt werden.

Ein weiterer Vorteil von PDO gegenüber anderen Projekten ist die Geschwindigkeit, da es vollständig in C++ programmiert wurde.

(vgl. [Pop07, S. 5])

18 http://www.oracle.com/technetwork/java/index-jsp-135919.html
19 http://pear.php.net/package/MDB2
20 http://adodb.sourceforge.net/
21 bei MDB2 2.5.0b5 war die letzte Veröffentlichung am 2012-10-29; bei ADOdb am 2012-09-04

2.3.2 *Limitierungen*

Bei einer Abstraktion wird stets etwas Spezifisches, durch das Weglassen von Details, in etwas Allgemeines überführt. Im Fall von PDO wird der andere Weg gegangen – es werden allgemeine SQL-Anfragen in den Dialekt[22] des Herstellers übersetzt.

Aus diesem Grund vermag es PDO nicht eine SQL-Abfrage, die in dem Dialekt eines Herstellers formuliert wurde, in den eines anderen zu übersetzen. Stattdessen muss die Anfrage so nah wie möglich am Standard gestellt werden, um als portabel zu gelten.

[Beispiel einfügen SQL-Standard und eigene Erweiterung von MySQL -> Siehe Buch Seite 17]

2.3.3 *Verwendung*

Der grundlegenden Unterschied von PHP-Extensions wie die für MySQL und PosgreSQL zu PDO besteht darin, dass die zuerst genannten eine prozeduale Bibliothek darstellen und PDO streng Objekt-orientiert aufgebaut ist. Während diese Extensions lediglich Funktionen zur Interaktion mit der Datenbank zur Verfügung stellen, die keine weiteren Informationen über den inneren Zustand der Verbindung besitzen, führt PDO Klassen ein, die sowohl die Verbindung als auch die eine Abfrage kapseln und als Schnittstelle dienen.

Die Klasse `PDO` beinhaltet die Verbindung zur Datenbank und stellt Methoden zum Verbindungsmanament bereit, während die Klasse `PDOStatement` eine Schnittstelle zu Anfragen und teilweise auch zur Ergebnismenge[23] bietet.

Im Folgenden wird die Verwendung der Klassen an einfachen Beispielen erläutert. Dabei werden die Unterschiede zur den klassischen Verfahren (MySQL und PostgreSQL) demonstriert. Damit PDO mit verschiedenen DBMS benutzt werden kann, müssen die Treiber des entsprechenden DBMS installiert sein. Als Grundlage der Abfragen und Ergebnisse dient eine Datenbank mit den folgenden Tabellen. Es wird davon ausgegangen, dass sie bereits erstellt wurde.

22 Als Dialekt wird die Hersteller-eigene Implementation des SQL-Standards genannt, dass sich im Umfang und Syntax vom Standard unterscheidet.

23 Da alle betrachteten Datenbanken den relationalen Datenbanken zuzuschreiben sind, handelt es sich bei den Datenbanktabellen um die mathematischen Beschreibung einer Relation . Zudem kann eine Relation/Tabelle als eine Menge aufgefasst werden. Verknüpft man Relationen mit Operatoren (Abfrage) erhält man stets wieder eine Relation. Somit ist das Ergebnis einer Abfrage eine Menge - die Ergebnismenge

```
students
+----+------------+-----------+-------+
| id | first_name | last_name | house |
+----+------------+-----------+-------+
|  1 | Lucius     | Malfoy    |   4   |
|  3 | Herminone  | Granger   |   1   |
|  4 | Ronald     | Weasley   |   1   |
|  5 | Luna       | Lovegood  |   3   |
|  6 | Cedric     | Diggory   |   2   |
+----+------------+-----------+-------+

houses
+----+------------+
| id | name       |
+----+------------+
|  1 | Gryffindor |
|  2 | Huffelpuff |
|  3 | Ravenclaw  |
|  4 | Slytherin  |
+----+------------+
```

2.3.4 *Verbindung aufbauen*

Traditionell wird eine Verbindung wie folgt aufgebaut:

```php
// MySQLi
$connection = mysqli_connect(
    $host,
    $username,
    $password,
    $dbname
);

// PostgreSQL
$connection = pg_connect(
    'host=' . $host .
    ' dbname=' . $dbname .
    ' user=' . $userName .
    ' password=' . $password
);
```

Dieses Beispiel zeigt, dass MySQLi[24] und PostgreSQL zwar einfache Methoden zur Verfügung stellen, diese sich jedoch in der Benutzung voneinander unterscheiden.

Um eine Verbindung mit PDO zu etablieren wird der Konstruktor verwendet, welcher ein Objekt vom Typ PDO zurückgibt. Die Signatur des Konstruktors sieht wie folgt aus:

24 MySQLi bietet sowohl eine prodezuale als auch eine objektorientierte API an. Da TYPO3 CMS ausschließlich den prodezualen Ansatz nutzt und sich dieser nahezu mit derAPI von MySQL deckt, sind die Bespiele in prozedualer Form gehalten.

```
public PDO::__construct(
  string $dsn
  [, string $username]
  [, string $password]
  [, array $driver_options]
)
```

Während die Parameter `$username` und `$password` selbsterklärend sind und über den letzten Parameter `$driver_options` Datenbanktreiber-spezifische Einstellungen übergeben werden können, erfordert der erste Parameter eine nähere Beschreibung.

Mit dem Kürzel `$dsn` (engl. Data Source Name) ist die Datenquelle gemeint, die in einem bestimmten Format übergeben werden muss. Dabei wird zuerst der Typ des DBMS angegeben und - getrennt von einem Doppelpunkt - der Datenbank-spezifische Teil.

```
// MySQL
$connection = new PDO(
  'mysql:host=$host;dbname=$db',
  $user,
  $pass
);

// PostgreSQL
$connection = new PDO(
  'pgsql:host=$host dbname=$db',
  $user,
  $pass
);
```

Das in der Variablen `$connection` enthaltene Verbindungsobjekt stellt den Ausgangspunkt für alles Weitere dar. Die naheliegendsten Aktionen nach dem Verbindungsaufbau sind das Absetzen einer SQL-Anfrage an die Datenbank und die Ausgabe des Ergebnisses.

2.3.5 *Datenbankabfragen*

Die als Beispiel dienende Abfrage soll die Nachnamen aller Studierenden in alphabetischer Reihenfolge ausgeben. Die in einer Variablen gespeicherten SQL-Abfrage wird an die Datenbank gesendet und das Ergebnis über eine Schleife ausgegeben. Zunächst wird der althergebrachte Weg gezeigt. Beide PHP-Extensions stellen dafür `*_query` Funktionen zur Verfügung, die eine Kennung der Datenbankverbindung zurückgeben. Im Fall eines Fehlers geben sie `FALSE` zurück.

```
$sql = 'SELECT last_name FROM students ORDER BY last_name';
// For MySQLi:
$result = mysqli_query($connection, $query);
while($row = mysqli_fetch_assoc($result)) {
    echo $row['last_name'] . ' ';
}

// PostgreSQL:
$result = pg_query($query);
while($row = pg_fetch_assoc($result)) {
    echo $row['last_name'] . ' ';
}
```

Das PDO-Objekt bietet dafür die Methode `query()` an, die ein Objekt vom Typ `PDOStatement` zurückgibt. Dieses implementiert das Interface Traversable und kann somit - analog zu einem Array - in einer Schleife durchlaufen werden. Ein Aufruf von `mysqli_result::fetch_assoc` erübrigt sich.

```
1  $sql = 'SELECT last_name FROM students ORDER BY last_name';
2
3  $statement = $connection->query($sql);
4
5  foreach($statement as $row) {
6    echo $row['last_name'] . ' ';
7  }
```

Die Ausgabe aller Bespiele lautet:

```
Diggory Granger Lovegood Malfoy Weasly
```

Um das Ergebnis der Abfrage sinnvoll nutzen zu können, gibt es verschiedene Stile (engl. fetch styles) in die es formatiert werden kann. Um dennoch die interne Struktur der Ergebnismenge beinflussen zu können, gibt es in PDO Konstanten, die als optionales Argument an `query()` übergeben werden. In der Defaulteinstellung benutzt PDO die Konstante `PDO::FETCH_BOTH`, bei dem das Ergebnis zum einen über den Spaltenbezeichner (wie im obigen Beispiel) als auch über eine Indexzahl angesprochen werden kann. Im Beispiel würde das so aussehen: `echo $row[0]`.

Zu allen `*_fetch_*`-Methoden gibt es das entsprechende Äquivalent als PDO-Konstante. Die Wichtigsten sind:

- `PDO::FETCH_ASSOC` - entspricht `*_fetch_assoc()`
- `PDO::FETCH_NUM` - entspricht `*_fetch_array()`
- `PDO::FETCH_ROW` - entspricht `*_fetch_row()`

Darüberhinaus definiert PDO noch weitere Konstanten, die keine Ensprechungen haben.

- `PDO::FETCH_OBJ` - liefert jede Zeile der Ergebnisrelation als Objekt zurück. Die Spaltenbezeichner werden dabei zu Eigenschaften der Klasse.
- `PDO::FETCH_LAZY` - wie `PDO::FETCH_OBJ`. Das Objekt wird jedoch erst dann erstellt, wenn darauf zugegriffen wird.
- `PDO::FETCH_CLASS` - liefert eine neue Instanz der angeforderten Klasse zurück. Die Spaltenbezeichner werden dabei zu Eigenschaften der Klasse.
- `PDO::FETCH_COLUMN` - liefert nur eine Spalte aus der Ergebnismenge zurück.

Dies stellt eine nicht abschließende Aufzählung dar. Die Dokumentation von PDO benennt weitere sogenannte *Fetch Styles*-Konstanten[25], die für diese Arbeit jedoch nicht von Interesse sind. Die Benutzung der Konstanten erfolgt per Übergabe als Parameter an die `query()`-Methode:

25 http://mx2.php.net/manual/en/pdo.constants.php

```php
$statement = $connection->query($sql, PDO::FETCH_NUM);

foreach($statement as $row) {
  echo $row[0];
}
```

Über das `PDOStatement`[26] werden weitere Möglichkeiten wie die Methoden `fetch()` und `fetchAll()` angeboten, um das Ergebnis zu erhalten. Diese Methoden müssen auch genutzt werden, wenn statt der `foreach`-Schleife eine `while`-Schleife genutzt werden soll:

```php
$statement = $connection->query($sql);

while($row = $statement->fetch(PDO::FETCH_ASSOC)) {
  echo $row['last_name'];
}
```

Zudem gibt es mit `fetch_column()` und `fetch_Object()` Alternativen für die Verwendung von `fetch()` in Verbindung mit den entsprechenden Konstanten.

2.3.6 *Prepared Statements*

Bereits MySQLi führt Prepared Statements ein, somit sind sie in der PHP-Welt nicht so neu. Während MySQLi nur einen Typ von Prepared Statements unterstützt, bietet PDO eine weitere sinnvolle Variante an. Im Folgenden wird das Konzept und der Nutzen von Prepared Statements kurz erklärt.

Prepared Statements können als eine Vorlage für SQL-Abfragen verstanden werden, die immer wieder, mit verschiedenen Werten, ausgeführt werden. Dabei kann ein DBMS die Struktur dieses Templates einmalig analysieren und vorkompiliert im Cache speichern. Bei jedem erneuten Aufruf setzt es lediglich die anderen Werte anstelle von Platzhaltern ein, was die Ausführung schneller macht. (vgl. [Pop07, S. 75])

Zur Demonstration soll je ein Codebeispiel dienen. Dabei fügen wir neue Studierende in die oben gezeigte Datenbanktabelle ein. Der sprechende Hut[27] hat bereits über die Häuser der Neuzugänge entschieden. Um die Abfrage einfach zu halten, wird der Fremdschlüssel der Tabelle für die Häuser direkt in dem Query angegeben.

Die Daten der Studierenden liegen in einem assoziativen Array vor und können somit über eine For-Schleife durchiteriert werden. Pro Schleifendurchlauf wird ein Studierender der Datenbank hinzugefügt. Die Werte werden mit der `pdo::quote()`-Methode maskiert um SQL-Injections[28] zu unterbinden.

26 Leider ist der Begriff dieser Klasse etwas unglücklich gewählt oder es ist ein Designfehler von PDO, denn ein Objekt dieser Klasse repräsentiert zum einen ein (Prepared) Statement und, nachdem die Anfrage ausgeführt wurde, die Ergebnisrelation. Die Methoden der Klasse agieren somit einmal auf dem Statement und einmal auf dem Ergebnis.

27 http://de.harry-potter.wikia.com/wiki/Sprechender_Hut

28 SQL-Injections werden im Kapitel 2.3.7 behandelt.

```php
$students = array (
    array (
        'last_name' => 'Ellesmere',
        'first_name' => 'Corin',
        'house' => 1
    ),
    array (
        'last_name' => 'Tugwood',
        'first_name' => 'Havelock',
        'house' => 4
    ),
    array (
        'last_name' => 'Fenetre',
        'first_name' => 'Valentine',
        'house' => 3
    )
)

foreach ($students as $student) {
    $sql = 'INSERT INTO students (last_name, first_name, house)
      VALUES (' . $connection->quote($student['last_name']) .
        ',' . $connection->quote($student['first_name']) .
        ',' . $connection->quote($student['house']) . ')';

    $connection->query($sql);
}
```

Bei jedem Durchlauf wird eine neue Abfrage mit den aktuellen Daten erzeugt und an die Datenbank geschickt. In diesen Fall bietet sich die Benutzung von Prepared Statements an, da sich pro Iteration lediglich die Werte ändern.

```php
$statement = $connection->prepare(
  'INSERT INTO students (last_name, first_name, house)
    VALUES (?, ?, ?)');

foreach ($students as $student) {
    $statement->execute(
      array(
        $student['last_name'],
        $student['first_name'],
        $student['house']);
    );
}
```

Die hier, anstelle der eigentlichen Daten, verwendeten Fragezeichen stellen Platzhalter dar, die als *Positional Placeholders* (engl. Positions Platzhalter) bezeichnet werden. Die Daten werden der Methode `PDOStatement::execute()` in einem Array übergeben. Dabei ist die Reihenfolge wichtig, da ansonsten die Daten in die falschen Spalten der Tabelle geschrieben werden.

Bei der Benutzung von Prepared Statements kann auf die Maskierung per `PDO::quote()` verzichtet werden, da dies die Datenbank übernimmt.

PDO bietet - im Gegensatz zu MySQLi - mit den *Named Paramentern* noch eine weitere Möglichkeit für Platzhalter an. Anstelle von Fragezeichen werden Bezeichner mit einem vorangestellen Doppelpunkt verwendet. Der Vorteil von dieser Variante, dass die Reihenfolge bei der Übergabe der Daten an die `PDOStatement::execute()`-Methode kei-

ne Rolle mehr spielt. Das folgende Listing zeigt den gleichen Code von oben jedoch diesmal mit Named Parametern. Die Daten werden dieses Mal als Key/Value-Paar übergeben, bei dem der Key den benannten Platzhalter darstellt und der Value die einzufügenden Daten.

```
$statement = $connection->prepare(
  'INSERT INTO students (last_name, first_name, house)
    VALUES (:lastname, :firstname, :house)');

foreach ($students as $student) {
    $statement->execute(
      array(
        ':firstname' => $student['first_name'],
        ':lastname'  => $student['last_name'],
        ':house'     => $student['house']);
    );
}
```

Nun spielt die Reihenfolge keine Rolle mehr – die Daten werden in die richtige Spalten eingefügt.

Die Zuordnung einer Variablen zu einem Platzhalter wird *Binding* genannt; gebundene Variablen werden demzufolge als *Bounded Variables* bezeichnet. Neben der gezeigten Bindung über
`PDOStatement::execute()` bietet PDO spezialisiserte Methoden an, was folgende Ursachen hat:

1. Bei der gezeigten Bindung werden die Variablen stets als String behandelt. Es ist nicht möglich dem DBMS mitzuteilen, das der übergebene Wert einem anderen Datentyp entspricht.

2. Die Variablen werden bei dieser Methode stets als In-Parameter übergeben. Auf den Wert der Variablen kann innerhalb der Funktion nur lesenend zugegiffen werden. Man nennt diese Übergabe auch *by Value*. Es gibt jedoch Szenarien in denen der Wert der Variable innerhalb der Funktion geändert werden soll. Dann müssen die Parameter als Referenz (*by Reference*) übergeben werden und agieren als In/Out-Paramaeter. Einige DBMS unterstützen dieses Vorgehen und speichern das Ergebnis der Abfrage wieder in der übergebenen Variable.

Das Äquivalent zum obigen Beispiel ist die Methode `PDOStatement::bindValue()`, bei der die Variable als In-Parameter übergeben wird. Für jeden zu bindenden Platzhalter muß die Methode aufgerufen werden, die den Name des Platzhalters, den zu bindenen Wert und die optionale Angabe des Datentyps erwartet. PDO bietet dazu vordefinierte Konstanten an, die den SQL Datentypen entsprechen.

```
$statement = $connection->prepare(
  'INSERT INTO students (last_name, first_name, house)
    VALUES (:lastname, :firstname, :house)');

foreach ($students as $student) {
    $statement->bindValue(':lastname', $student['first_name']);
    $statement->bindValue(':firstname', $student['last_name']);
    $statement->bindValue(':house', $student['house'], PDO::PARAM_INT);

    $statement->execute();
}
```

Die Methode zur Übergabe der zu bindenden Werte per Referenz heißt `PDOStatement::bindParam()`. Die Funktionsweise unterscheidet sich dahingehend von `bindValue()`, als dass die Werte, welche in der Variablen gespeichert sind, erst dann aus der Adresse im Speicher ausgelesen werden, wenn `execute()` ausgeführt wird, während `bindValue()` die Werte sofort bei Aufruf der Funktion ausliest. Aus diesem Grund muß `bindValue()` innerhalb der Schleife stehen.

```
$statement = $connection->prepare(
  'INSERT INTO students (last_name, first_name, house)
    VALUES (:lastname, :firstname, :house)');

$statement->bindParam(':lastname', $student['first_name']);
$statement->bindParam(':firstname', $student['last_name']);
$statement->bindParam(':house', $student['house'], PDO::PARAM_INT);

foreach ($students as $student) {
    $statement->execute();
}
```

2.3.7 *SQL-Injections*

[SQL injections Mummy image einfügen]

Bei SQL-Injections kann über das Frontend einer Anwendung eine Zeichenkette in eine SQL-Abfrage injiziert werden, die die Fähigkeit besitzt den betroffenen SQL-Code derart zu verändern, dass er

- Informationen wie den Adminbenutzer der Webanwendung zurückliefert
- Daten in der Datenbank manipuliert um ein neuer Adminbenutzer anzugelegen
- oder die Datenbank ganz- oder teilweise löscht

Für ein kurzes Beispiel einer SQL-Injection soll ein Formular dienen, indem nach den Nachnamen der Studierenden aus Hogwarts gesucht werden kann. Der gesuchte Datensatz wird ausgegeben wenn er gefungen wird, ansonsten erscheint eine entsprechende Meldung. Der in das Inputfeld eingegebene Wert wird von PHP automatisch in der Variablen `$_REQUEST` gespeichert und kann in der Anwendung ausgelesen werden. `'SELECT * FROM students WHERE last_name = Diggory'` stellt eine mögliche, zu erwartende SQL-Anfrage dar.

[Balsamico Formular einfügen]

```
$sql = "SELECT * FROM students
  WHERE last_name = '" . $_REQUEST['lastName'] . "'";

$statement = $connection->query($sql);

foreach($statement as $student) {
  echo 'Lastname: ' . $student['last_name'] . "\n";
  echo 'Firstname: ' . $student['first_name'] . "\n";
  echo 'Haus: ' . $student['house'] . "\n";
}
```

Dieser Code beinhaltet zwei Fehler:

1. Es wird nicht überprüft, ob `$_REQUEST['lastName']` leer ist oder was ganz anders enthält als erwartet.

2. die Benutzereingabe wird nicht maskiert

Im Falle einer leeren Variable, sähe die Abfrage so aus: `'SELECT * FROM students WHERE last_name = '`. Im besten Fall gibt sie eine leere Ergebnismenge zurück im schlechtesten einen Fehler. Diese Problem ist leicht zu lösen, indem zum einen auf die Existenz der Variablen geprüft wird und zum anderen ob sie einen Wert enthält. Zusätzlich sollte noch auf den Datentyp des enthaltenen Wertes geprüft werden. Erst dann wird die Anfrage abgesetzt.

Da die Eingabe nicht maskiert wird, interpretiert der SQL-Parser einige Zeichen als Teil als Steuerzeichen der SQL-Syntax. Beispiele solcher Zeichen sind das Semikolon, der Apostroph und ein Backslash.

Selbst ein Websitebesucher ohne böse Ansichten könnte mit der Suche nach einem Studenten mit dem Namen O'Hara die SQL-Injection auslösen. Die in diesen Fall an die Datenbank gesendetet SQL-Anfrage `'SELECT * FROM students WHERE last_name = 'O'Hara';` würde wohl einen Fehler auslösen, da der Parser die Anfrage nach dem *O* anhand des Apostroph als beendet interpretiert und *Hara* kein gültiges Sprachkonstrukt von SQL darstellt.

Ein Angreifer könnte hingegen die Eingabe in das Formular nach `' or '1'='1` verändern. Damit würde sich diese Abfrage `'SELECT * FROM students WHERE last_name = '' or '1'='1';` ergeben.

Wird die Maskierung mit einer entsprechenden Prüfung von Benutzereingaben kombinert, verhindert das die Gefahr von SQL-Injections – eine richtige Anwendung vorrausgesetzt. Wie in Fußnote [28] bereits erwähnt wurde, müssen an `pdo::query()` übergebene SQL-Anfragen mit `PDO::quote()` maskiert werden. Traditionell wird dafür die PHP-Methode `addslashes()` oder die jeweiligen Maskierungsmethoden der DBMS verwendet. Für MySQLi wird `mysqli_real_escape_string()` verwendet.

Werden Prepared Statements genutzt, müssen die Benutzereingaben trotzdem überprüft, jedoch nicht mehr maskiert werden. Dies übernimmt dann die Datenbank. Wird dem Prepared Statement noch der Typ des Wertes mitgeteilt, kann das DBMS eine Typprüfung vornehmen und ggf. einen Fehler zurückgeben.

2.4 *Arbeitsweise*

Aufgrund seiner Aufgabe stellt der Protoyp einen tiefen Eingriff in die Architektur von TYPO3 dar. Vor diesem Hintergrund ist es umungänglich, dass er alle Anforderungen erfüllt, die an eine Systemextension gestellt werden, auch wenn er keine Systemextension ist.

Dies fängt mit der Einhaltung der TYPO3 Coding Guidelines an, geht über die Versionierung des Codes hin zum Einbinden in das TYPO3 Testframework

2.4.1 *Formatierung des Quellcodes*

Programmier neigen zu unterschiedlichen Programmierstilen, was solange kein Problem darstellt, solange sie allein an einem Projekt arbeiten. Kommen mehr Entwickler

hinzu und es vermischen sich verschiedene Stile, kann es schnell unübersichtlich werden. Um den Sinn eines Programms zu verstehen, hilft es allmein, wenn die Codebasis in einer konsistenten Form formatiert wurde. Dies erhöht die Wartbarkeit und verbessert die Code Qualität.

Coding Guidelines stellen dabei das Regelwerk dar auf das sich die Entwickler eines Projekts geeingt haben.

In den TYPO3 Coding Guidelines (CGL)[29] wird die Verzeichnisstruktur von TYPO3 selbst, sowie die von Extensions erläutert. Sie erklären die Namenskonventionen für Dateien, Klassen, Methoden und Variablen und beschreiben den Aufbau einer Klassendatei mit notwendigen Inhalt, der unabhängig von dem Zweck der Klasse vorhanden sein muss.

Der größter Teil der CGL behandelt die Formatierung verschiedene Sprachkonstrukte wie Schleifen, Arrays und Bedingungen.

Da das Regelwerk mit 28 Seiten recht umfangreich aussfällt, wird zur Überprüfung des Quellcodes das Programm PHP_CodeSniffer[30] in Zusammenhang mit dem entsprechendem Regelset für das TYPO3 Projekt [31].

2.4.2 *Unit Testing*

Eine Anforderung an den Prototyp war, dass er die alte Datenbank API weiterhin unterstützt. Dadurch ist gewährleistet, dass noch nicht angepasste Extensions weiterhin funktionieren. Um dies zu überprüfen muß die Funktionaliät von TYPO3 in Verbindung alter API / neue API fortlaufend getestet werden.

Ein möglicher Ansatz wäre es, mit zwei identischen TYPO3 Installationen zu starten, die mit der Zeit auseinander divergieren, indem eine der beiden APIs immer mehr auf die neue API umgebaut würde. Das Testen dabei kann jedoch nur auf eine stichprobenartige Überprüfung der Funktionaliät des manipulierten Systems beruhen, da es nahezu unmöglich ist, durch dieses manuelle Vorgehen alle Testfälle abzudecken.

Ein andere Ansatz würde auf der Code Ebene ansetzen und pro Methode der alten API verschiedene Testfälle definieren, welche nachvollziehbar wären und immer wieder ausgeführt werden könnten.

Das dafür in Frage kommende Framework heißt PHPUnit[32]. Es stellt das PHP Pedant von dem aus der Javawelt bekannten JUnit dar und wird von TYPO3 unterstützt. TYPO3 bringt selbst schon über 6000 UnitTests mit[33].

Das eingangs umrissene Szenario stellt lediglich ein greifbares Beispiel für den Nutzen von Unit Tests dar und ist keineswegs auf Spezialfälle wie Refactorings oder dem Austausch einer API beschränkt. In der vorliegenden Arbeit wurden alle implementierten Methoden der neuen API mit Unit Tests - in Verbindung mit PHPUnit -abgedeckt.

Um die Tests zu starten gibt es mehre Möglichkeiten:

29 http://docs.typo3.org/typo3cms/CodingGuidelinesReference/6.2/_pdf/manual.t3cgl-6.2.pdf
30 Link
31 Link
32 LINK
33 https://travis-ci.org/TYPO3/TYPO3.CMS/builds/23070563

Im TYPO3 Backend

[Bild einfügen]

Auf der Konsole

[Bild einfügen]

In der IDE

[Bild einfügen]

2.4.3 *Versionsverwaltung*

[Bild einfügen] Als Versionsverwaltung wurde GIT[34] in Verbindung mit dem Code Hostingdienst GiHub[35] genutzt. Github dient zum einen als Backup im Falle eines Festplattenausfalls und zum anderen als späterer Anlaufpunkt der Extension für Interessierte.

GitHub bietet zudem auch eine Aufgabenverwaltung[36] an, dass bei dem Projekt mehr oder minder intensiv genutzt wurde, dem jedoch nach der Veröffentlich als OpenSource Software mehr Bedeutung zukommen wird.

Ferner hat sich um GitHub eine Vielzahl von Services entwickelt, welche unter anderen dabei hilft, die CodeQualität zu analysieren. Die zwei zu erwähnenden Projekte sind Travis[37] und Scrutinizer[38], welche für das Projekt genutzt wurden.

2.4.4 *Travis-CI*

[Bild einfügen] Travis-CI ist ein Continuous Integration[39] (CI) Service, welcher auf Github gehostete baut.

Der Begriff "bauen" kommt von Sprachen wie C oder Java, die erst kompiliert werden müssen (engl. to compile oder to build), bevor sie ein ausführbares Programm darstellen. Im Gegesatz zu interpretierten Sprachen, die zur Laufzeit von einem Interpreter übersetzt werden. Hier definiert der Begriff "bauen" im Zusammmenhang mit CI das auschecken des Codes aus einem Repository und die Ausführung von:

- Tests (Unit-, Smoke-, Akteptanz- und Behaviortests),

34 http://git-scm.com/
35 http://github.com
36 Issuetracking
37 LINK
38 LINK
39 Kontinuierliche Integration

- statischer Codeanalysen wie den oben genannten PHP_CodeSniffer, PHPDepend[40], PHP Mess Detection[41] oder PHP Copy and Paste[42]
- oder das Verpacken der Software in ein Realease fähiges Format wie PHAR[43] oder einen Tarball[44].

Die Konfiguration von Travis-CI erfolgt über eine Textdatei im YAML-Format[45] und liegt im Wurzelverzeichnis des Projekts.

[Beispiel einer YAML Datei für Travis einfügen]

Da der Prototyp aus zwei Teilen besteht, wurden zwei Travisprojekte erstellt. Eins für die Extension und eins für den modifizierten TYPO3 Kern.

Konfiguration für die Extension

PHPUnit CodeSniffer PHPCPD PHPMessDetection PHPLOC

Ziel war eine möglichst 100% Testabdeckung zu erreichen.

Konfiguration für den TYPO3 Kern

Während bei den Tests der Extension auf eine innere Konsistenz des Programmcodes geschaut wurde, wurde bei den Tests des Kerns Augenmerk auf die Integration der Extension in das modifizierte TYPO3 gelegt. Ziel war es, dass alle mitgelieferten Tests des Cores erfüllt werden.

PHPUNit

2.4.5 *Scrutinizer*

[Bild einfügen] Es kann zweifelsfrei gesagt werden, dass Travis-CI das Arbeitstier ist. Außer den oben beschriebenen Tätigkeiten macht er nicht viel. Das Besondere daran ist jedoch, dass er diese Aufgaben unermüdlich mit der gleichen Gewissenhaft ausführt. Das ist die Stärke eines solchen Progamms. Am Ende steht jedoch immer nur ein binäres Ergebnis:

* die Tests sind bestanden * es sind Tests fehlgeschlagen

Anhand dieses Ergebnis können dann weitere Entscheidungen getroffen werden. Zum Beispiel wird es sinnvoll sein bei einem negativen Ergebnis darauf zu verzichten ein Download Paket zu erstellen und veröffentlichen.

Braucht man jedoch eine graphische Auswertung, der durch die statische Codeanalysen gewonnen Daten, kommt ein Service wie Scrutinizer zum Einsatz. Wie schon bei Travis-CI erfolgt die Konfiguration über eine YAML Datei.

40 LINK
41 LINK
42 LINK
43 LINK
44 LINK
45 LINK

Wie Tests gezeigt haben[46] ist es gegenwärtig nicht möglich den TYPO3 Kern mit Scrutinizer auszuwerten, da das Tool verschiedene Limits implementiert hat. Dies Limits werden von TYPO3 teilweise um ein Vielfaches überschritten. Es ist lediglich gelungen die Auswertung der Analyse zum Umfang des Codes und zur Copy and Paste Detection darzustellen. Da dies jedoch keine ausreichende Aussage über die Code Qualität treffen kann und die Qualität des Codes des TYPO3 Kerns nicht im Fokus dieser Arbeit liegt, wurde auf die Auswertung verzichtet.

Es wurde lediglich ein Srcutinizer Projekt für die Extension erstellt.

[Auswertung des Scrutinzer beschreiben]

Eins wird bei 10.000 Issues[47] überschritten, welches von fast alles Analysen übertroffen wird. Zum Beispiel und zum anderen hat es einen zeitlichen Timeout, der bei der Analyse der Ergebnisse wie zum Beispiel der Code Coverage[48] überschritten wird.

Der Grund für die Verbindung mit Scrutinizer war auch hier der Gedanke an die spätere Veröffentlichung der Extension und stellt die Grundsteinlegung für eine gleichbeleibend hohe Codequalität dar, die überprüfbar sein soll.

2.4.6 *IDE*

Als Editor wurde die Integrated Development Environment (IDE) PHPStorm.

[Bild einfügen]

PHPStorm verfügt über Autovervollständig von Variablen, Methoden und Klassen, unterstützt den Programmierer bei der Erstellung von Klassen durch Templates und bietet vielseitige Integration von externen Programmen.

So ist es zum Beispiel möglich die PHPUnit Tests von PHPStorm aus zu starten und mit ein wenig Konfiguraton**web:kowalke14** kann man sich die CodeCoverage im Editor anzeigen lassen.

[Bild einfügen]

Besonders hilfreich ist es jedoch sich die Ergebnisse des PHP_CodeSniffers anzeigen zu lassen um zumindest beim Feinschliff den Code GCL konform zu formatieren.

Die überaus leistungsfähige Integraton von XDebug runden die Funktionaliät, die der Editor bildet, ab.

Mit Hilfe von PHPStorm war es möglich sich schnell und umfassend in den Quellcode von TYPO3 und der Extension DBAL einzuarbeiten

46 https://scrutinizer-ci.com/g/Konafets/TYPO3v4-Core/inspections
47 Mit Issues sind Stellen gemeint, die laut Scrutinizer überarbeitet werden müssten
48 Abdeckung der Codebasis mit Unit Tests

3

PROTOTYPISCHER NACHWEIS DER HERSTELLBARKEIT

3.1 *Refactoring der alten Datenbank API*

3.1.1 *Tests für die alte Datenbank API*

3.2 *Testgetriebene Implementierung der neuen Datenbank API*

- Adapter - alte API Klasse erbt von neuer API Klasse - Verbindung mit der Datenbank via Doctrine - Methodenübername wenn sinnvoll - Neue Methoden

3.2.1 *Einführung von Query Objekten*

SELECT

INSERT

UPDATE

DELETE

TRUNCATE

3.3 *Anwendung der neuen Datenbank API*

3.4 *Überprüfen der Funktionalität*

4

AUSBLICK

- Einladung zum ACME14N
- Einbau in den Core
- Nutzung von Doctrine Migrations
- Nutzung von Doctrine ORM
- Umbau der Datenbankobjekts zu einem Datenbankconnection Pool Design Pattern Static Fabric oder Fabric

ZUSAMMENFASSUNG

[Ada95] Douglas Adams. *The Hitchhiker's Guide to the Galaxy*. Englisch. Auflage: Reissue. Valley View, Calif.: Del Rey, Sep. 1995. isbn: 9780345391803.

[Dam10] Karsten Dambekalns. *FrOSCamp 2010 in Zürich*. Englisch. Sep. 2010. url: http://karsten.dambekalns.de/blog/froscamp-2010-in-zurich.html (besucht am 08. 04. 2014).

[DEM14] Karsten Dambekalns, Benjamin Eberlei und Christian Müller. *The reasons why TYPO3 Flow switched to Doctrine ORM*. microblog. Apr. 2014. url: https://twitter.com/konafets/status/453102477081341952 (besucht am 08. 04. 2014).

[DRB08] Dmitry Dulepov, Ingo Renner und Dhiraj Bellani. *TYPO3 extension development developer's guide to creating feature-rich extensions using the TYPO3 API*. English. Birmingham, U.K.: Packt Pub., 2008. isbn: 9781847192134 1847192130 1847192122 9781847192127.

[Ebe12] Benjamin Eberlei. *Extension that replaces TYPO3 Extbase ORM with Doctrine2*. Englisch. Apr. 2012. url: https://github.com/simplethings/typo3-extbase-doctrine2-extension (besucht am 09. 04. 2014).

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. en. Addison-Wesley Professional, 2003. isbn: 9780321127426.

[Lab+06] Kai Laborenz u. a. *TYPO3 4.0: das Handbuch für Entwickler ; [eigene Extensions programmieren ; TypoScript professionell einsetzen ; barrierefreie Websites, Internationalisierung und Performancesteigerung ; inkl. AJAX und TemplaVoila]*. German. Bonn: Galileo Press, 2006. isbn: 9783898428125 3898428125.

[Mar12] Thomas Maroschik. *WIP: Doctrine DBAL integration*. Englisch. Dez. 2012. url: https://git.typo3.org/Packages/TYPO3.CMS.git/tree/b0a64733d2de2396c1f91dd559b37f0a4b652766 (besucht am 09. 04. 2014).

[Pop07] Dennis Popel. *Learning PHP Data Objects: A Beginner's Guide to PHP Data Objects, Database Connection Abstraction Library for PHP 5*. Packt Publishing Ltd, 2007.

[T3N09] T3N. *Jonathan Wage über seinen Einstieg bei Doctrine und die Zukunft des Projekts: "Wir hatten gute Entwickler, aber keine Vision"*. Dez. 2009. url: http://t3n.de/magazin/jonathan-wage-seinen-einstieg-doctrine-zukunft-projekts-224058/ (besucht am 01. 05. 2014).

[TYP] TYPO3 Association. *What is TCA?* Englisch. url: http://docs.typo3.org/typo3cms/TCAReference/WhatIsTca/Index.html (besucht am 24. 04. 2014).

[TYP08] TYPO3 Association. *Berlin Manifesto*. Englisch. Okt. 2008. url: http://typo3.org/roadmap/berlin-manifesto/ (besucht am 06. 04. 2014).

[TYP13] TYPO3 Core Team Mailingliste. *TYPO3 - Team Core - RFC: Integrate Doctrine2 into TYPO3 CMS*. Englisch. Jan. 2013. url: http://typo3.3.n7.nabble.com/RFC-Integrate-Doctrine2-into-TYPO3-CMS-td237490.html (besucht am 09. 04. 2014).

API Application Programming Interface

BE Backend

CMF Content Management Framework

CMS Content-Management-System

DBAL Database Abstraction Layer

DBMS Database Management System

ECMS Enterprise Content Management-System

EM Extension Manager

FE Frontend

GPL2 GNU General Public License v.2

IDE Integrated Development Environment

JCR Content Repository for Java Technology API

JDO Java Data Objects

MVC Model-View-Controller

ORM Object-relational mapping

PDO PHP Data Objects

PHP PHP: Hypertext Processor

SQL Structured Query Language

T3Assoc TYPO3 Association

TCA Table Content Array

TER TYPO3 Extension Repository

WCMS Web Content Management-System

TABELLENVERZEICHNIS

ABBILDUNGSVERZEICHNIS

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Flensburg, den 4. Mai 2014

_____

Stefan Kowalke