Berliner Hochschule für Technik

Fachbereich VI - Informatik und Medien
Studiengang Medieninformatik Online
Luxemburger Str. 10
13353 Berlin

Konzeption und Implementierung einer TYPO3-Extension zur Analyse und Darstellung von statistischen Daten gegebener Content Management Systeme

Bachelorarbeit zur Erlangung des Grades eines Bachelor of Science (B.Sc.)

- Bachelorarbeit -

eingereicht von
Alexander Ullrich
Matrikel-Nr.: 770481

Betreuer:
Sebastian Kreideweiß
Prof. Dr. Tramberend

26. Dezember 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In dieser Arbeit soll die Konzeption, Entwicklung und Implementierung einer TYPO3 Extension zur Analyse und Darstellung von statistischen Daten gegebener Content Management Systeme durchgeführt und anschaulich vorgestellt werden. Die entwickelte Extension soll anschließend im Onlinebetrieb produktiv zur Anwendung kommen und helfen, auf Grundlage der statistischen Daten über die Verteilung von benutzten CMS in einer bestimmten Kategorie entsprechende Entscheidungen zu treffen.

1.1 Motivation

Um digitale Inhalte einer Webseite unabhängig vom Design, der verwendeten Technologie, sowie dem technischen Aufbau gemeinsam erstellen, bearbeiten und organisieren zu können, wurden sogenannte **"CMS - Content Management System"**[1] als Softwarelösung entwickelt.

Wie beim Onlineanbieter **"CMS Matrix"** [2] ersichtlich, gibt es heutzutage mehr als 1300 verschiedene CMS, welche die unterschiedlichsten Ansätze in Technologie, Komplexität, der Größe der Community, sowie der Weiterentwicklung und dem Support des bestehenden Systems beinhalten und verfolgen. Es wird schnell erkennbar, daß man nicht jedes CMS in seiner Funktionalität beherrschen und kennen kann, was die Auswahl des richtigen CMS für den eigenen Anwendungsfall erschwehrt.

Da verschiedene CMS für die unterschiedlichsten Anwendungsfälle entwickelt worden sind, dort auch ihren Schwerpunkt finden und diese CMS bereits auf online Plattformen stabil laufen, wäre die Information über das jeweils eingesetzte CMS eine große Entscheidungshilfe bei der Suche nach dem geeigneten System für den eigenen Anwendungsfall.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird eine TYPO3 Extension als Prototyp entwickelt, wobei die einzelnen Entwicklungsschritte und die eingesetzten Technologien dokumentiert und vorgestellt werden.

Die Extension wird für TYPO3 (Version 11) [3] erstellt und anschließend installier- und ausführbar sein. Sie wird die Möglichkeit zur Eingabe neuer Kategorien und URL's, sowie die visuelle Auswertung der statistischen Daten über die Verteilung von benutzten CMS einer Kategorie in Tabellen- und Chartform für Frontendbenutzer zur Verfügung stellen.

Die Grundlage für die Auswertung der statistischen Daten liefern die von WhatCMS.org [4] übertragenen Informationen, welche in der zu entwickelnden Extension über API Schnittstelle oder eine entsprechende Datei im JSON Format importiert werden können.

[1] Englisch für "Inhaltsverwaltungssystem" [1]

1.3 Struktur der Arbeit

Diese Arbeit gliedert sich in sechs Kapitel, welche die Entwicklung des Prototyp einer TYPO3 Extension von der Idee bis zur einsatzfähigen Anwendung beschreiben.

Nachdem in der Einleitung Informationen zur Motivation und Zielsetzung benannt worden, wird im zweiten Kapitel auf den fachlichen und technischen Hintergrund der Anwendung eingegangen. Besonderes Augenmerk fällt dabei auf die Quelle der Daten, sowie die Beschreibung der zugrunde liegenden Schnittstelle.

Im dritten Kapitel der Arbeit wird auf die funktionalen und technischen Anforderungen der Anwendung eingegangen. Es wird neben "Mehrsprachigkeit" und "Sicherheit" auch die "Mindestanforderung" an das jeweilige Grundsystem beschrieben.

Das vierte Kapitel widmet sich dem theoretischen Teil der Arbeit. Es wird in seiner ausführlichen Beschreibung die Architektur und technische Konzeption des bekannten "Domain-Driven Design", inklusive der Betrachtung aller einzelnen Bausteine beinhalten.

Ab dem fünften Kapitel wird die Umsetzung des praktischen Teils der Arbeit, vom Grundkonzept über das Domänenmodell bis hin zum fertigen Code der Anwendung, mit dem kompletten Entwicklungszyklus beschrieben und entsprechend vorgestellt.

Der abschließende Teil wird sich im sechsten Kapitel mit dem kritischen Rückblick auf das gesamte Projekt beschäftigen. Mit Fazit und Ausblick werden gesammelte Erkenntnisse und Ideen für die zukünftige Weiterentwicklung des TYPO3 Extension Prototyps aufgeführt.

2 Fachlicher und Technischer Hintergrund

Im folgenden wird die Quelle der zugrundeliegenden Daten zur statistischen Erhebung, die verwendete Technologie zum Austausch der Daten, sowie der Grundgedanke zur Idee und Festlegung des Namen für die Extension beschrieben.

2.1 WhatCMS.org - Wer ist das?

WhatCMS.org [5] beschreibt sich selbst als Anbieter, welcher Anfragen von Benutzern aus der ganzen Welt bearbeitet, die mehr über die von ihnen verwendeten Webseiten erfahren möchten.

Um diese Informationen so genau wie möglich zur Verfügung stellen zu können, verwendet WhatCMS.org eigene Erkennungsalgorithmen und hat eine Reihe von Qualitätskontrollsystemen eingerichtet.

Ein besonderes Markenzeichen des von WhatCMS.org verwendeten Erkennungsalgorithmus ist die Fähigkeit, Content Management Systeme pro Seite, statt pro Domain zu erkennen.

2.2 WhatCMS.org - Was kann man damit tun?

Nachdem man sich als Nutzer auf der Webseite registriert und anschließend angemeldet hat, besteht die Möglichkeit sogenannte "Batches" anzulegen. Eine Batch ist eine Art Kategorie (siehe Abbildung 1), welcher man die verschiedensten URL [6] zuweisen kann, um anschließend durch den Erkennungsalgorithmus von WhatCMS.org das dahinter laufende CMS und die jeweilige Technologie ermitteln zu lassen.

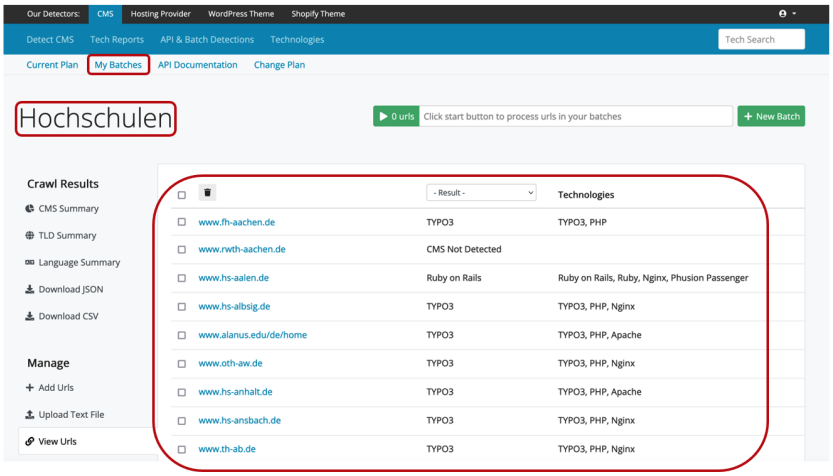

Abbildung 1: WhatCMS.org - Screenshot Batch Hochschulen

Sobald der Erkennungsalgorithmus alle CMS einer Batch erkannt hat, besteht die Möglichkeit die Daten in verschiedenen Dateiformaten zu exportieren oder auch visuell in einem Verteilungschart (siehe Abbildung 2) anzeigen zu lassen.

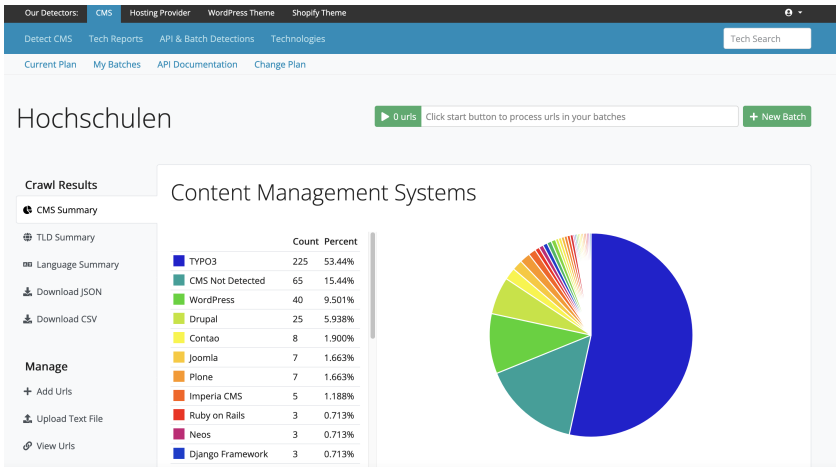

Abbildung 2: WhatCMS.org - Screenshot Chart Hochschulen

2.3 WhatCMS.org - API Schnittstelle

Neben der Visualisierung der Daten und der Erkennung des jeweiligen CMS, sowie der jeweiligen Technologie hinter einer URL bietet WhatCMS.org noch die Möglichkeit der Abfrage über eine API Schnittstelle.

2.3.1 API Schnittstelle - Anfrage

Wie in der Dokumentation [7] von WhatCMS.org beschrieben, wird die Anfrage über einen GET Request an den API Server von WhatCMS.org gestellt. Die Abfrage URL des API Server hat dabei der Struktur wie in Abbildung 3 ersichtlich zu erfolgen. Zuerst kommt die URL des API Server, gefolgt vom privaten API Schlüssel und der abschließenden URL, für welche man das CMS ermitteln möchte.

Abbildung 3: WhatCMS.org - Aufbau API Request URL

2.3.2 API Schnittstelle - Antwort

Wurde die Anfrage erfolgreich vom API Server entgegengenommen und bearbeitet, so erhällt man eine Antwort im JSON [8] Format mit allen wichtigen Informationen zur angefragten URL zurück (siehe Abbildung 4). Diese strukturierten Daten können nun ausgewertet und weiterverarbeitet werden.


```
{
    "request": "https://whatcms.org/API/CMS?key=YOUR-API-KEY&url=en.wikipedia.org",
    "request_web": "https://whatcms.org/?s=en.wikipedia.org",
    "result": {
        "code": 200,
        "msg": "CMS Found",
        "id": 8,
        "name": "MediaWiki",
        "confidence": "high",
        "cms_url": "https://whatcms.org/c/8_MediaWiki"
    },
    "private": false
}
```


Abbildung 4: WhatCMS.org - JSON Antwort API Request

2.4 CMScensus - Grundgedanke zur Idee

Möchte man die Ergebnisse seiner WhatCMS.org Batches gern der breiten Öffentlichkeit zur Analyse und Auswertung zugänglich machen, so ist dies nur möglich, wenn man seine privaten Zugangsdaten von WhatCMS.org ebenfalls öffentlich macht.

Eine weitere Schwierigkeit liegt in der Tatsache begründet, daß jeder öffentliche Besucher andere Interessen hat und ein wildes Bearbeiten der Batches früher oder später zu großem Chaos führt.

CMScensus wird eine TYPO3 Extension, welche dieses Problem auf elegante Art und Weise löst, indem es die Daten von WhatCMS.org verwendet, weiter verarbeitet und auf jeder beliebigen TYPO3 Website darstellen kann. Desweiteren wird CMScensus die Möglichkeit der Vorschläge neuer Kategorien und URLs durch entsprechende Frontendbenutzer beinhalten und damit die Funktionalität erweitern.

2.5 CMScensus - Namensgebung

Da Census für Volkszählung [9] steht und eine gesetzlich angeordnete Erhebung statistischer Bevölkerungsdaten ist, bot sich der Name zur Verwendung in Kombination mit CMS als **"CMScensus"** an und wird damit als Name für die neue TYPO3 Extension verwendet.

3 Anforderungen

Nachfolgend werden die Anforderungen an die TYPO3 Extension beschrieben, welche sich aus den im Vorfeld geführten Meetings mit Herrn Kreideweiß, in der Rolle als "Product Owner" von CMScensus ergaben. Sie wurden mit dem Gedanken erstellt, ein Minimal-Viable-Product[2] zu entwickeln, welches anschließend auch eine produktive Anwendung finden soll.

3.1 Funktionale Anforderungen

3.1.1 Mehrsprachigkeit

Da die Extension nicht nur im deutschsprachigen Raum, sondern auch international Anwendung finden soll, muss die Möglichkeit der Mehrsprachigkeit implementiert sein.

3.1.2 Vorschläge

Frontendbenutzer sollen Vorschläge für neue Kategorien und dazugehörige URLs unterbreiten können. Damit die vorgeschlagenen URLs der neuen Kategorie zugeordnet werden, muss diese nach absenden des Kategorievorschlages mit auswählbar sein. Bei Eingabe einer fehlerhaften URL soll der Nutzer durch eine entsprechende Warnung darauf hingewiesen werden.

3.1.3 Sicherheit

Der Administrator muss die Möglichkeit haben, die Extension so zu konfigurieren, daß neue Vorschläge nur von angemeldeten Frontendbenutzern erstellt werden können. Damit soll die Möglichkeit des SPAM, durch Eingabe unzähliger unnützer Kategorien und URLs von Spaßnutzern unterbunden werden.

3.1.4 Import

Der Import von den WhatCMS.org gelieferten Daten soll sowohl über eine Datei im JSON Format, als auch über eine Anfrage an die WhatCMS.org API Schnittstelle mit Rückgabe im JSON Format funktionieren.

3.1.5 Ausgabe

Der Redakteur hat die Möglichkeit für jede Kategorie eine Seite zu erstellen, welche die prozentuale Verteilung der CMS der einzelnen URLs visuell als Tabelle, sowie als Chart darstellt.

[2] In einem Minimal Viable Product (MVP) wird ein Minimalumfang für ein Produkt (z.B. eine Software) über eine Anzahl von "Musst-Have-Features" definiert. Wenn das Produkt weniger Funktionen als diese definierten hat, erwirtschaftet es keinen Nutzen. Folgend kann das MVP dann schrittweise um weitere Funktionen erweitert werden. [34]

3.2 Technische Anforderungen

3.2.1 Grundsystem

Da die Extension nach Fertigstellung auf einer öffentlichen Webseite innerhalb eines CMS laufen soll, wurde sich für TYPO3 als Grundsystem entschieden. Dabei soll die Entwicklung auf der neuesten TYPO3 (Version 11) durchgeführt werden. Ein wichtiger Grund für diese Entscheidung liegt unter anderem darin begründet, daß TYPO3 (Version 11) wie in Abbildung 5 ersichtlich durch den sog. **"Long Term Support"** (LTS) [10] bis 2027 unterstützt wird. Daraus ergibt sich für die Extension eine entsprechende Langlebigkeit.

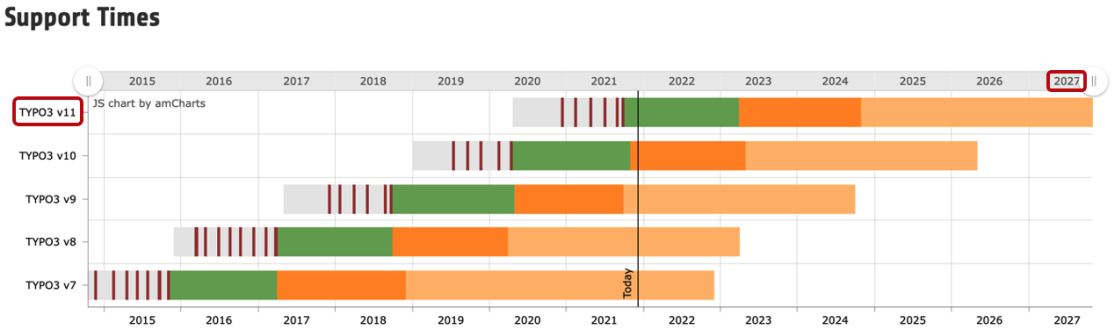

Abbildung 5: TYPO3 - Roadmap

Eine weitere wichtige Systemanforderung an die Extension beinhaltet, daß sie mindestens unter PHP7.4 [11] lauffähig und mit Composer v2 [12], sowie der TYPO3 Console [13] installierbar sein sollte.

Für die Versionierung soll das freie Open Source Kontrollsystem Git [14] zum Einsatz kommen. Das entsprechende "Git Repository" wird auf "GitHub" eingerichtet und ist unter der URL **"https://github.com/usadmin77/bachelorarbeit.git"** für ausgewählte Benutzer öffentlich erreichbar.

4 Architektur und technische Konzeption

Für die Umsetzung der Entwicklung der Extension wird das von Eric Evans in seinem 2003 veröffentlichten Buch "Domain-Driven Design: Tackling Complexity in the Heart of Software" [35] geprägte und vorgestellte **"Domain-Driven Design"** (DDD) verwendet.

4.1 Domain-Driven Design

Das Hauptziel der Verwendung liegt darin begründet, Probleme und Erwartungen des Kunden von Anfang an vollständig zu verstehen und die gesamte Bandbreite der jeweiligen Geschäftsaktivitäten in all seiner Komplexität als **"Domain"** zu bezeichnen und zu betrachten.

Wie in dem von Michael Schams im Jahr 2010 veröffentlichten Buch "TYPO3 Extbase: Moderne Extension-Entwicklung mit Extbase und Fluid" [36] (S. 35 ff.), basiert das Domain-Driven Design auf den folgenden zwei wichtigen Annahmen:

- Der Schwerpunkt des Softwaredesigns liegt auf der Sachlichkeit und der Fachlogik.
- Der Entwurf komplexer Zusammenhänge sollte auf einem Fachmodell basieren.

4.1.1 Das Domänenmodell

Es ist eine wichtige Anforderung an DDD, daß der Entwurf der Software über ein Model zu erfolgen hat [36] (S. 37). Ein Model ist dabei die vereinfachte Beschreibung der Wirklichkeit, was bedeutet, daß das Domänenmodell (engl. domain model) einen Plan für die darin enthaltenen Objekte, deren Eigenschaften und ihren Relationen untereinander darstellt.

4.1.2 Ubiquitous Language (UL)

Da die Sprache der Domäne die entscheidende Sprache ist, muss sie als zentrales Element beim Modellieren eindeutig festgelegt werden. Dies geschieht am besten über ein "Ubiquitous Language"[3] Glossar, welches zu Beginn der Projektphase durch alle Projektbeteiligten gemeinsam erstellt wird. Ziel ist es, daß alle Projektmitglieder genau wissen, worum es sich bei den verschiedensten Begriffen der Domäne handelt und diese von den einzelnen Mitgliedern nicht mehr falsch verstanden und interpretiert werden.

[3] Englisch für "allgegenwärtige Sprache"

4.2 Bausteine des DDD

Wie in Abbildung 6 ersichtlich, setzt sich das Domain-Driven Design aus den verschiedensten Bausteinen zusammen, welche nachfolgend detailliert vorgestellt und beschrieben werden.

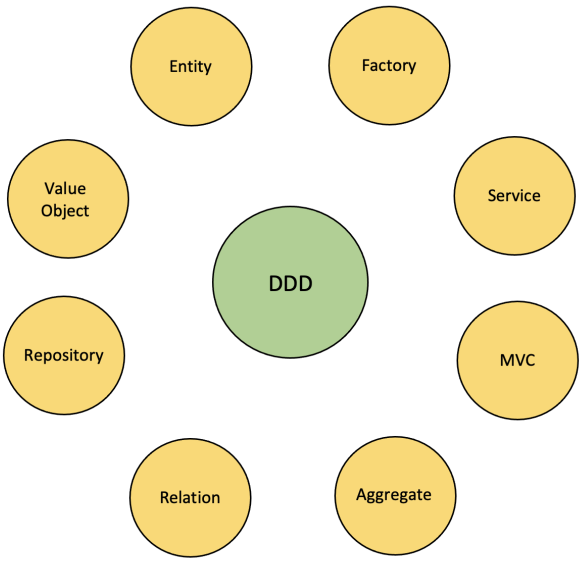

Abbildung 6: Bausteine des Domain-Driven Designs

4.2.1 Entity

Eine Entity (Entität) ist ein Domänenobjekt, welches eine globale und beständige Identität besitzt. Um es mit dem realen Leben zu vergleichen, könnte dies zum Beispiel die eindeutige Zuordnung einer Person sein, welche trotz Heirat oder Umzug immer die gleiche Nummer im Reisepass behält und darüber jederzeit identifiziert werden kann. Auch der Fingerabdruck ändert sich niemals und könnte in diesem Beispiel zur Bestimmung der Identität herangezogen werden.

4.2.2 Value Object

Ein Value Object wird durch die Summe der Eigenschaften definiert und hat im Gegensatz zur Entity keine globale Identität. Es wird über seine Eigenschaften angesprochen und ist immutable (nicht änderbar). Als Beispiel könnte hier die Farbe "Rot" als Value Object mit dem Farbcode "c00" genannt werden, welche nicht einfach im Namen in "Blau" geändert werden kann, ohne eine andere oder neue Farbe zu erhalten.

4.2.3 Service

Damit Entities und Value Objects als Grundbausteine des DDD untereinander kommunizieren können, existieren sogenannte Service-Klassen, welche auch mit dem Begriff **"Services"** bezeichnet werden. Als Beispiel aus der realen Welt kann man hier die Überweisung zwischen zwei Konto-Objekten nennen.

4.2.4 Factory

Ist die Erstellung der Domänobjekte selbst zu komplex (wenn z.B. Assoziationen notwendig sind, welche vom Domänobjekt nicht benötigt werden), oder soll die spezifische Erzeugung des Domänobjektes zur Laufzeit ausgetauscht werden, so kommen sogenannte **"Factories"** zum Einsatz. Sie dienen dazu, Domänobjekte als dezentrale, spezifische Factory-Objekte zu erstellen. Ein gutes Beispiel für den Einsatz einer Factory wäre die Instanziierung von vier Rad-Objekten im ersten Schritt, mit der anschließenden Verknüpfung des Auto-Objekts im zweiten Schritt und der abschließenden Rückgabe dieses Auto-Objektes an den jeweiligen Aufrufer.

4.2.5 Repository

Da Datenbanken und Speichermechanismen im DDD nicht existieren und DDD die Infrastruktur streng trennt, wurden sog. **"Repositories"** eingeführt. Ein Repository ist dabei als eine Art Blackbox zu betrachten, welches Objekte persistieren und lediglich die Suche nach Objekten ermöglichen kann. Es wird immer genau einem Objekt zugeordnet und erlaubt auch nur den Zugriff auf das entsprechende Objekt. Möchte man ein Objekt über ein anderes Repository ermitteln, so muss eine Beziehung zu dem gesuchten Objekt bestehen.

4.2.6 Aggregate

Ein Aggregate ist eine Zugriffskontrolle und wird im DDD als **"Aggregate Root"** bezeichnet. Wie ein Türsteher den Eingang eines Nachtclubs bewacht und entscheidet, welche Person Zutritt zu den Objekten (z.B. Tanzfläche oder alkoholische Getränke) innerhalb des Nachtclubs hat und wem der Zutritt verweigert wird, entscheidet der Aggregate Root über den Zugriff auf das jeweilige Objekt. In Extbase gibt es die klare Vorgabe, daß alle Objekte die "Aggregate Root" sind, ein Repository und einen Controller inklusive entsprechender Actions haben müssen.

4.2.7 Relation

Die Relation stellt eine Verbindung zwischen zwei Objekten dar. In Extbase gibt es dafür genau vier verschiedene **"Relationen"**, die sich wie folgt zusammensetzen:

1:1 Bei der **"1:1 Relation"** gibt es für ein Objekt nur genau ein anderes Objekt, mit dem eine Verbindung bestehen kann. Ein Beispiel aus der realen Welt wäre die Sozialversicherungsnummer, welche immer nur genau zu einer Person gehört.

1:n Bei der **"1:n Relation"** können von einem Objekt beliebig viele Verbindungen zu anderen Objekten bestehen. So hat zum Beispiel ein Blog beliebig viele Beiträge, wobei ein Beitrag immer nur genau zu einem Blog gehören kann.

n:1 Die **"n:1 Relation"** gibt an, daß es für beliebig viele Objekte genau ein Objekt gibt, mit dem eine Verbindung besteht. Als Beispiel kann der Beitrag eines Blogs genannt werden, welcher immer nur einen Author hat, wobei der Author im Gegensatz beliebig viele Beiträge schreiben kann.

m:n Bei der **"m:n Relation"** können von beliebig vielen Objekten Verbindungen zu beliebig vielen anderen Objekten bestehen. Ein gutes Beispiel dafür ist die Verwendung in CMScensus, wo einer Kategorie mehrere URLs zugewiesen werden können und im Gegensatz eine URL auch in mehreren Kategorien vorkommen kann.

4.2.8 MVC

MVC steht für das **"Model-View-Controller"** [15] Design Pattern und entspricht einer "Layered Architecture"[4], welche im Domain-Driven Design gefordert ist. Die Aufgaben der einzelnen Komponenten des MVC sind dabei wie folgt gegliedert:

M Model ("das Modell"):
Diese Ebene kann man auch als Domain ("Domäne") bezeichnen, da sie die komplette Geschäftslogik und die Geschäftsdaten beinhaltet.

V View ("die Ansicht"):
Diese Ebene wird als sog. Ansichts- oder Template-Ebene bezeichnet, da sie für die Ausgabe der Daten zuständig ist. Sie wird in Extbase durch Templates und die dahinterliegende Template Engine "Fluid" realisiert.

C Controller ("die Steuereinheit"):
Diese Ebene darf weder Geschäfts-, noch View-Logik enthalten und kümmert sich ausschließlich um die Steuerlogik. Extbase setzt hier auf das "Slim Controller" Konzept, wobei der Controller selbst in Actions unterteilt ist, welche die Steueraufgaben übernehmen.

[4] Englisch für "Mehrschichtige Architektur" [37]

5 Umsetzung

5.1 Grundkonzept

Damit die zu entwickelnde Extension dem Sinn und Zweck gerecht wird, sollte sie mindestens folgende Funktionen bereitstellen:

- Ein Frontendbenutzer kann eine unbegrenzte Anzahl von neuen Kategorien und URLs vorschlagen.
- Beim vorschlagen der Kategorie oder URL kann zu dieser eine Beschreibung hinzugefügt werden.
- Jede vorgeschlagene Kategorie oder URL wird automatisch als Vorschlag markiert.
- Wird eine URL vorgeschlagen, so können zu dieser alle zur Verfügung stehenden Kategorien ausgewählt werden.
- Jede URL kann für die automatische Abfrage bei WhatCMS.org sowohl aktiviert, wie auch deaktiviert werden, wobei sie standardmäßig erst einmal deaktiviert ist.
- Neben dem URL Namen und der Beschreibung muss außerdem die Möglichkeit der Speicherung des verwendeten CMS gegeben sein.

5.2 Das Glossar

Damit alle Projektbeteiligten das gleiche Verständnis für die verwendete Terminologie haben, wird das in Abbildung 7 ersichtliche Glossar erstellt und für die Domänenobjekte die englische Sprache benutzt.

Terminologie	Deutsche Übersetzung	Beschreibung	Relevant für das Model
Batch	Charge	ist die bei WhatCMS.org verwendete Bezeichnung für die Sammlung aller URLs in einer bestimmten Kategorie.	Nein
Category	Kategorie	entspricht der Batch von WhatCMS.org und beinhaltet alle URLs, welche zu dieser Kategorie zusammengefasst werden sollen.	Ja
URL	URL	Standard für die Adressierung einer Website im World Wide Web.	Ja
Description	Beschreibung	Beschreibender Text zur Verwendung des jeweiligen Objektes.	Ja
WhatCMSType	WhatCMS Type	ist die von WhatCMS.org verwendete CMS Bezeichnung.	Ja
Proposal	Vorschlag	wird zur Markierung einer Kategorie oder URL als entsprechender Vorschlag verwendet.	Ja
Auto Update	Automatisches Update	wird zur Markierung einer URL für das ermitteln des dazugehörigen CMS beim nächsten Import Lauf zu WhatCMS.org verwendet.	Ja

Abbildung 7: Glossar - CMScensus

5.3 Domain Model ("Domänenmodell")

Auf Grundlage des Glossar wird nun das Domänenmodell erstellt. Wie in Abbildung 8 ersichtlich, ergeben sich daraus die folgenden Details:

- Es gibt zwei Domänenobjekte: *Url* und *Category*, welche vom Typ *Entity* sind (erkennbar an dem Buchstaben E in der rechten oberen Ecke).
- Bei beiden Objekten handelt es sich um *Aggregate Root* (erkennbar an den Buchstaben AR in der rechten unteren Ecke).
- Das Domänenobjekt Url besitzt die sieben Eigenschaften ("properties"): *name, description, is_proposal, only_next_auto_update, is_auto_update_planned, every_update und whatcmstype.*
- Das Domänenobjekt Category hat die Eigenschaften: *name, description und is_proposal.*
- Zwischen beiden Domänenobjekten besteht eine *m:n - Beziehung*, was so viel bedeutet wie, daß eine Url eine beliebige Anzahl von Categories enthalten kann. Außerdem kann einer Category eine beliebige Anzahl von Urls zugewiesen werden.

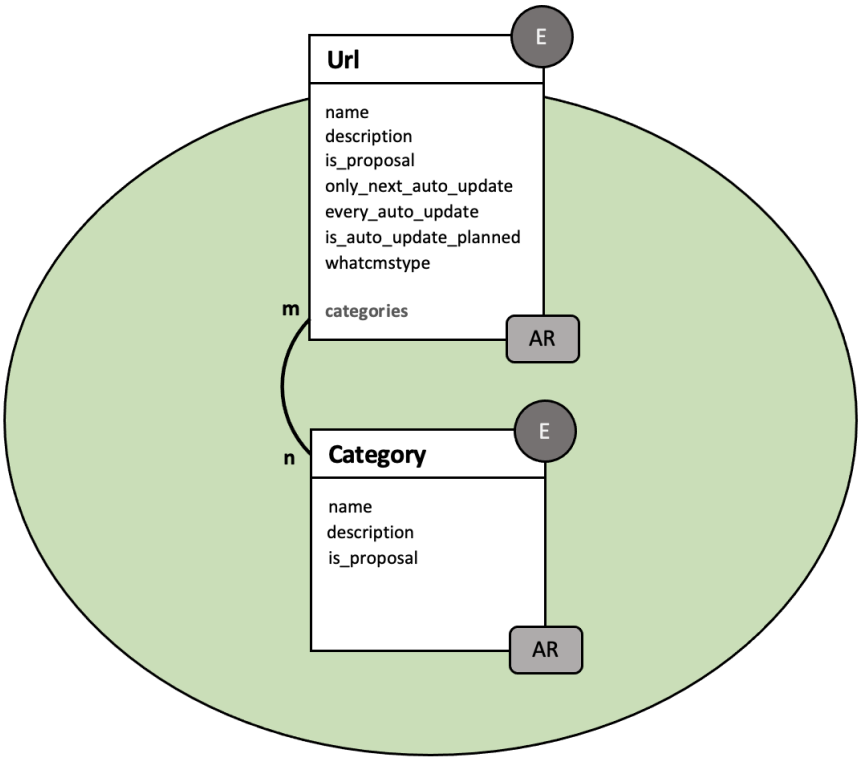

Abbildung 8: Domänenmodell - CMScensus

5.4 Vorbereitung (”Entwicklungsumgebung’)

Um mit der Entwicklung der Extension beginnen zu können, wird in diesem Abschnitt die Einrichtung der Entwicklungsumgebung in kurzer Form beschrieben. Es wird angenommen, daß auf dem Entwicklungssystem PHP 7.4 [11], Composer v2 [12], DDEV [16] sowie Docker [17] installiert sind und in lauffähiger Form vorliegen.

5.4.1 TYPO3 v11 installieren

Wie in der Dokumentation [18] von TYPO3.org ausführlich beschrieben, wird im ersten Schritt der Ordner erstellt, in welchem unsere neue TYPO3 Installation liegen soll. Dazu wird die Shell geöffnet und folgender Befehl eingegeben:

```
mkdir TYPO3v11AUBA
```

nun wechseln wir in den eben erstellten Ordner durch Eingabe von:

```
cd TYPO3v11AUBA
```

und erzeugen das entsprechende DDEV TYPO3 Projekt mit nachfolgendem Befehl:

```
ddev config  --project-type=typo3 --docroot=public --create-docroot
```

Da es festgelegte Anforderungen an das System gibt, müssen diese in der **config.yaml** Datei, welche im **Ordner .ddev** zu finden ist, so wie in Abbildung 9 ersichtlich eingestellt und angepasst werden.

```
1  name: TYPO3v11AUBA
2  type: typo3
3  docroot: public
4  php_version: "7.4"
5  webserver_type: nginx-fpm
6  router_http_port: "80"
7  router_https_port: "443"
8  xdebug_enabled: false
9  additional_hostnames: []
10 additional_fqdns: []
11 mariadb_version: "10.2"
12 mysql_version: ""
13 provider: default
14 use_dns_when_possible: true
15 composer_version: "2"
```

Abbildung 9: CMScensus - DDEV config.yaml

Um mit der Einrichtung fortzufahren, wird der DDEV Container anschließend durch Eingabe des folgenden Befehls gestartet:

```
ddev start
```

Nach erfolgreichem Start des DDEV Container sollte so wie in Abbildung 10 ersichtlich die URL angezeigt werden, welche später in die Browser Addresszeile zum Starten der Installation eingefügt werden muss.

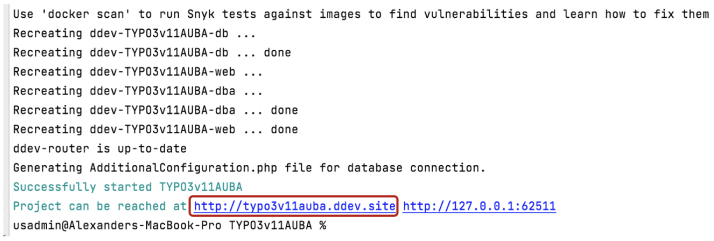

Abbildung 10: CMScensus - ddev start

Nun wird die komplette "TYPO3 v11 CMS Base Distribution" via Composer geholt und alle Abhängigkeiten durch Eingabe des nachfolgenden Befehls aufgelöst:

```
ddev composer create 'typo3/cms-base-distribution:^11'
```

Bevor mit der Installation von TYPO3 über den Browser begonnen werden kann, muss noch die leere Datei "FIRST_INSTALL" im public Ordner durch Eingabe des touch Befehls erstellt werden:

```
ddev exec touch public/FIRST_INSTALL
```

Wurden alle Schritte erfolgreich durchgeführt, so sollte nun die Installation von TYPO3 über den Browser möglich sein und wie in Abbildung 11 zu sehen, durch Eingabe der beim Start generierten URL in die Browser Adresszeile zum ersten Installationsschritt führen.

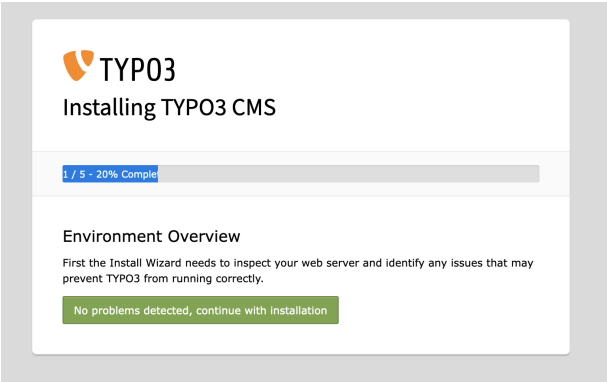

Abbildung 11: CMScensus - TYPO3 Installation Schritt 1

Da die restlichen Schritte der Installation selbsterklärend sind, wird auf die weitere Erläuterung im Rahmen dieser Arbeit verzichtet.

5.4.2 Extension Builder installieren

Der **"Extension Builder"** [19] ist selbst eine Extension, welche die Entwicklung einer neuen eigenen Extension erleichtern soll. Er bietet nach der Installation die Möglichkeit, im Backend via GUI[5] seine eigene Extension vorzubereiten und ein erstes kleines Gründerüst automatisch erstellen zu lassen.

Um den Extension Builder via Composer in die aktuelle TYPO3 Instanz zu installieren, wird der nachfolgende Befehl in die Shell im Hauptverzeichnis der TYPO3 Installation eingegeben:

```
ddev composer req friendsoftypo3/extension-builder
```

Nach erfolgreicher Installation sollte nun wie in Abbildung 12 zu sehen der Extension Builder im Backend in dem Modulbereich der **"Admin Tools"** sicht- und auswählbar sein.

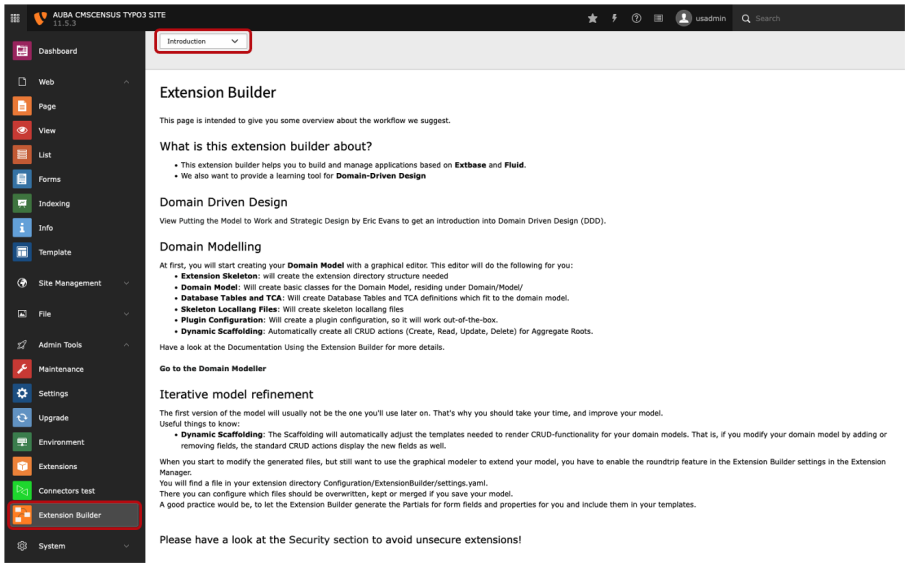

Abbildung 12: TYPO3 - Extension Builder

5.4.3 Introduction Package installieren

Damit die visuelle Ausgabe einer lauffähigen TYPO3 Webseite schneller realisiert werden kann, wird das offizielle **"TYPO3 Introduction Package"** [21] installiert. Wurde die Installation erfolgreich abgeschlossen, so findet man eine komplette TYPO3 Beispiel Webseite vor, in welcher anschließend eigene Änderungen vorgenommen werden können.

[5] Graphical User Interface - Englisch für "Grafische Benutzeroberfläche" [20]

Um die Installation des TYPO3 Introduction Package via Composer zu starten, muss der folgende Befehl ausgeführt werden:

```
ddev composer req typo3/cms-introduction
```

Damit die Einrichtung durchgeführt wird, ist anschließend folgender Befehl auszuführen:

```
ddev typo3cms extension:setup
```

Wurde die Installation erfolgreich abgeschlossen, so ist der Seitenbaum im Backend wie in Abbildung 13 unter dem Modulbereich **"Web"** im Modul **"Page"** ersichtlich.

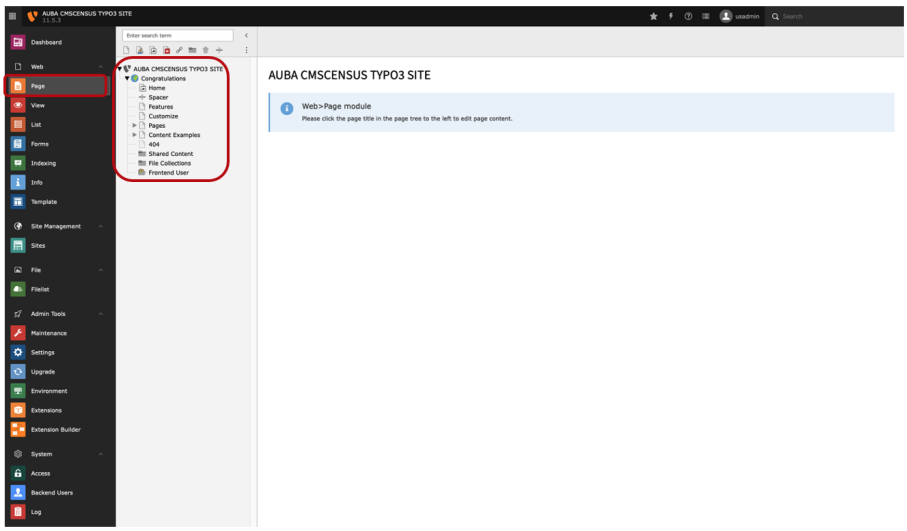

Abbildung 13: TYPO3 - Introduction Package

5.4.4 External Data Import installieren

Da eine Anforderung an die CMScensus Extension der Import von Daten im JSON Format ist und man bei der Entwicklung von Anwendungen auf die Wiederverwendung von bereits existierender Funktionalität zur Vermeidung von Redundanzen[6] achten sollte, wird an dieser Stelle für die Entwicklung der CMScensus Extension die bereits existierende Extension **"External Data Import"** [23] verwendet und an die eigenen Anforderungen durch entsprechende Erweiterungen angepasst.

[6] von lateinisch redundare, „überlaufen, sich reichlich ergießen“ [22]

Für die Installation der Extension "External Data Import" ist der nachfolgende Befehl einzugeben und auszuführen:

```
ddev composer req cobweb/external_import
```

Da die Extension den Scheduler von TYPO3 benötigt, muss auch hier zur abschließenden Einrichtung der folgende Befehl ausgeführt werden:

```
ddev typo3cms extension:setup
```

Nach erfolgreichem Abschluß der Installation, sollten jetzt wie in Abbildung 14 zu sehen, die Module **"Connectors test"**, **"Data Import"** und **"Log"** im Backend sichtbar geworden sein.

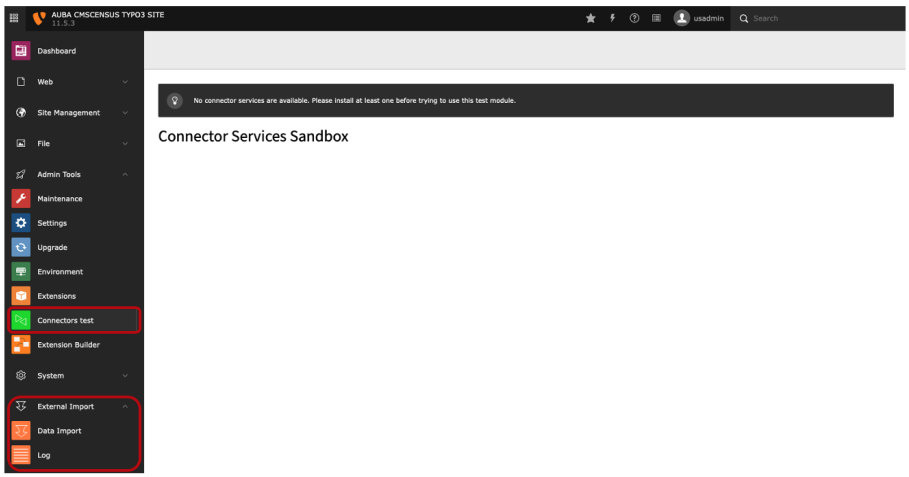

Abbildung 14: TYPO3 - External Data Import

5.4.5 Cache leeren

Eine gute Vorgehensweise ist das regelmäßige leeren des kompletten Cache der TYPO3 Installation, um eventuelle Artefakte vom System selbständig wegräumen zu lassen. Besonders nach der Installation neuer Module oder Extensions bietet sich das leeren des Cache an.

Damit man das leeren des Cache über die Konsole startet, ist nachfolgender Befehl in der Shell einzugeben:

```
ddev typo3cms cache:flush
```

5.5 Entwicklung der Extension

In diesem Abschnitt wird die komplette Entwicklung der CMScensus Extension beschrieben. Es wird sowohl die technische, als auch die konzeptionelle Umsetzung in der Beschreibung zu finden sein.

5.5.1 Erstellung Grundgerüst

Im ersten Schritt der Entwicklung wird das Grundgerüst der CMScensus Extension mit Hilfe des "Extension Builder" erstellt. Wie in Abbildung 15 zu sehen, werden zuerst alle Informationen der zu erstellenden Extension in die Konfigurationsfelder des Extension Builder eingetragen und anschließend abgespeichert.

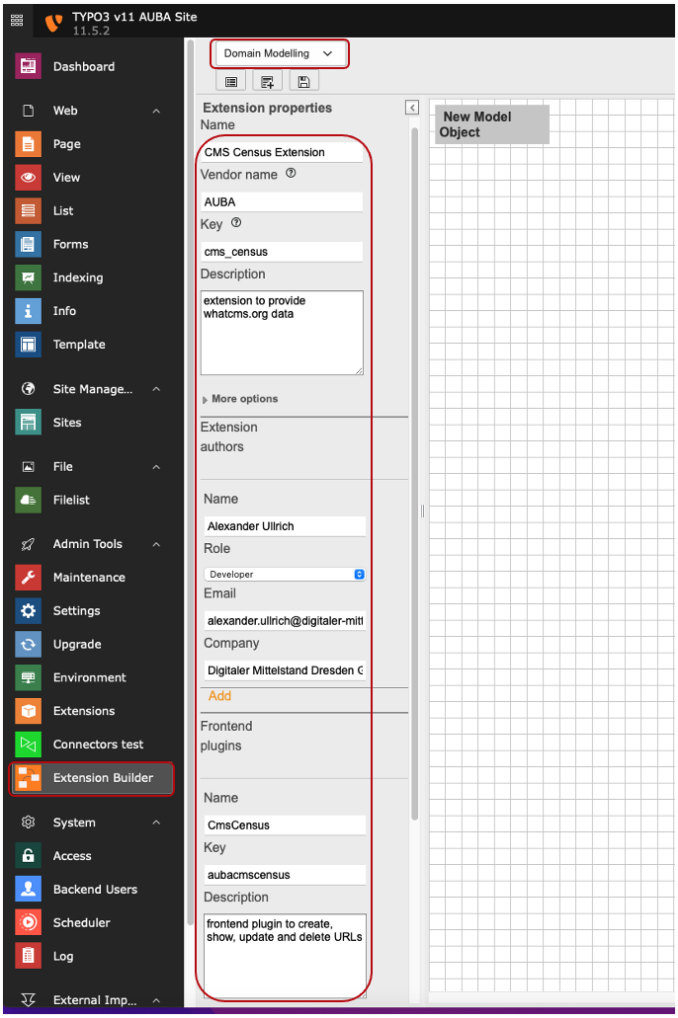

Abbildung 15: TYPO3 - Extension Builder new Model

Im nächsten Schritt wird das aus **Abschnitt 5.3** (Seite 13) theoretisch erstellte Domänenmodell, so wie in Abbildung 16 ersichtlich, modelliert und mit den entsprechenden Eigenschaften versehen.

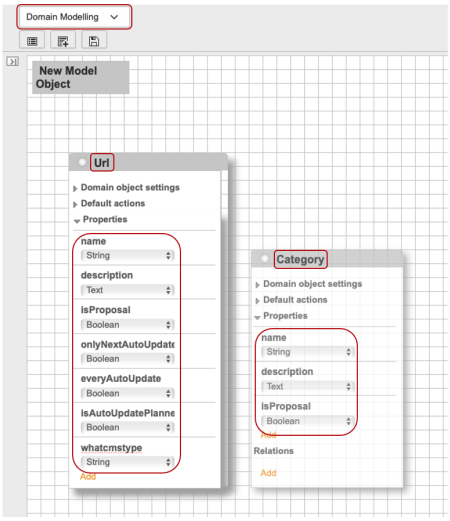

Abbildung 16: TYPO3 - Extension Builder Domänenmodell

Im letzten Schritt der Modellierung werden die beiden Objekte "URL und Category" in ihrer entsprechend dem Domänenmodell vorgesehenen **"m:n - Relation"**, so wie in der Abbildung 17 erkennbar, miteinander verbunden.

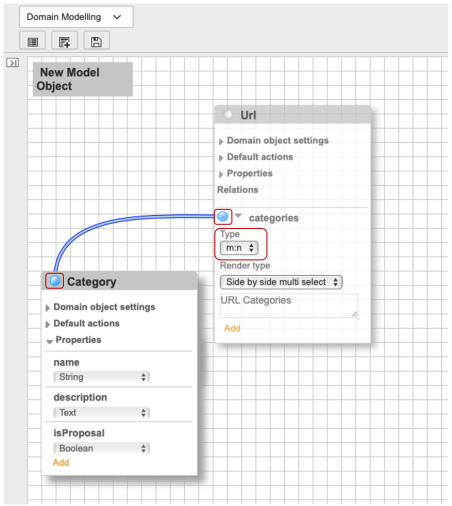

Abbildung 17: TYPO3 - Extension Builder Relation

Nachdem die Modellierung abgeschlossen ist, muss die neue Extension gespeichert werden, wobei auch ein komplettes Grundgerüst durch den Extension Builder erstellt wird.

Bevor dies jedoch geschieht, wird mit den nachfolgenden Befehlen in der Shell erst ein neuer Ordner mit dem Namen "packages" angelegt und anschließend in der Konfigurationsdatei "composer.json" ein Link zum Ordnerpfad hinzugefügt:

```
mkdir -p packages &&
ddev composer config repositories.local path "packages/*"
```

Nun kann die neue Extension abgespeichert und in die lauffähige TYPO3 Installation durch den folgenden Befehl entsprechend eingebunden werden:

```
ddev composer require auba/cms-census:@dev
```

Um zu überprüfen ob die Installation und Einbindung der Extension erfolgreich durchgeführt wurde, kann man sich wie in Abbildung 18 ersichtlich unter dem entsprechenden Modul **"Extensions"** alle lokal installierten Extensions anzeigen lassen und sollte damit auch die soeben installierte CMScensus Extension finden.

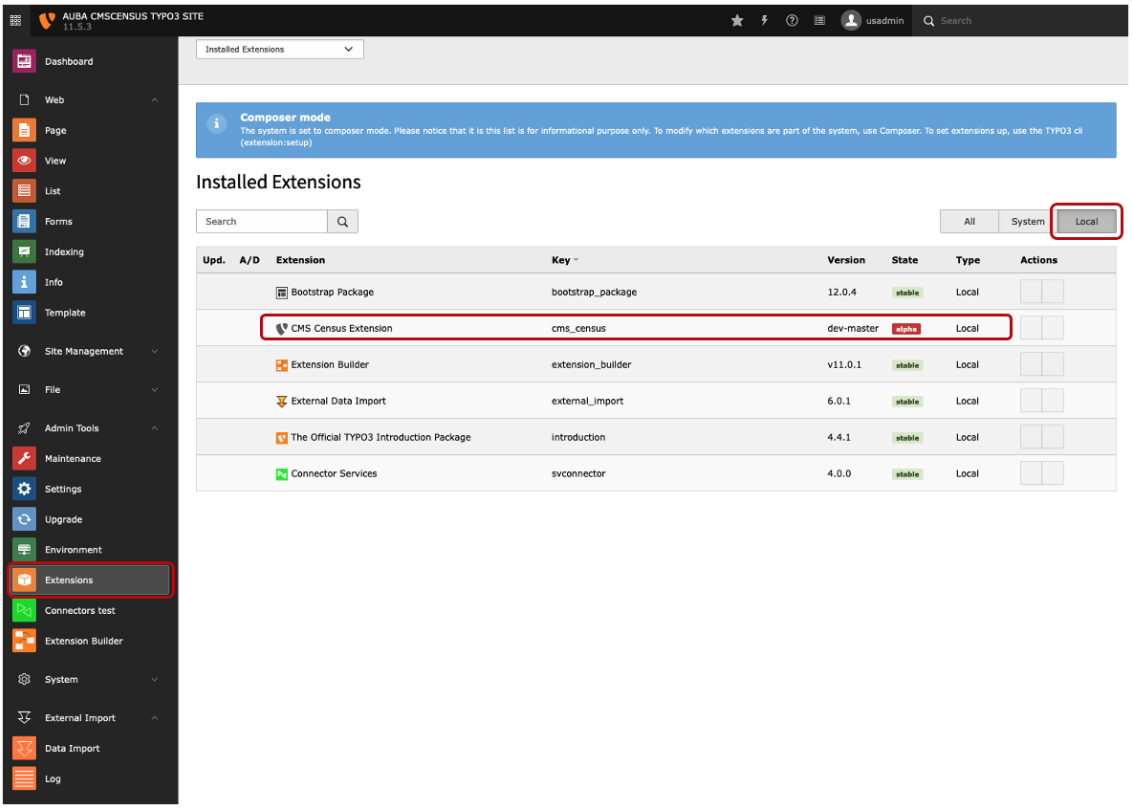

Abbildung 18: TYPO3 - lokal installierte Extensions

5.5.2 Erstellung Datei Import

Damit zur weiteren Entwicklung entsprechende Daten verfügbar sind, wird an dieser Stelle mit der Entwicklung des Datei Import begonnen. Dazu wird die Extension **"External Data Import"**, so wie in **Abschnitt 5.4.4** (Seite 17) bereits beschrieben verwendet und um benötigte Funktionalitäten erweitert.

Um den Datei Import zu realisieren, ist ein entsprechender Dateiablageort von nöten. Auf diesen hat der jeweilige Dateizugriff beim ausführen des Importszenario zu erfolgen. Er wird über das TYPO3 Backend für die CMScensus Extension, wie in Abbildung 19 zu sehen, durch anlegen der "ext_conf_template.txt" Datei konfigurier- und speicherbar gemacht. Neben dem **Dateipfad**, wird der **Dateiname** und die **Kategorie**, welcher der Inhalt zugewiesen werden soll, abgefragt und später unter den entsprechenden Variablennamen in der Datenbank gespeichert.

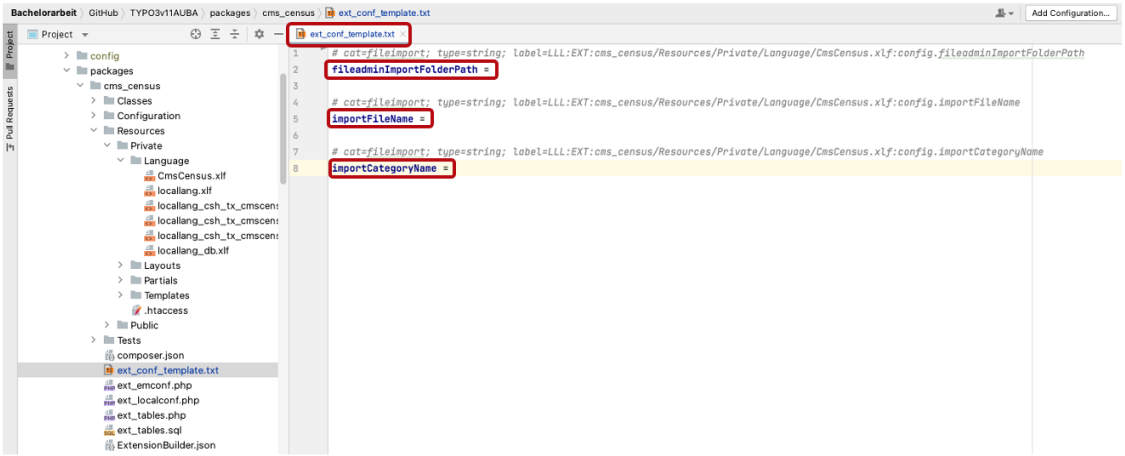

Abbildung 19: CMScensus - ext_conf_template.txt

Wie in Abbildung 20 ersichtlich, weisen die Kommentarinhalte über den Variablen in der "ext_conf_template.txt" Datei TYPO3 an, in welchem "Tab" sie in den "Settings" für die CMScensus Extension angezeigt werden sollen und wie die entsprechende Übersetzung der jeweiligen Ausgabetexte durch die Verknüpfung zur "CMSCensus.xlf" Datei lautet. Es wird außerdem der Typ definiert, welchen die jeweilige Variable hat. Später kann der Zugriff auf den Inhalt der Variablen in der Applikation über die **"ExtensionConfiguration::class"** der **"TYPO3\CMS\Core\Configuration\ExtensionConfiguration"** durchgeführt werden.

Durch diesen Zugriff können die entsprechenden Werte zur Auswertung und Programmsteuerung in den jeweiligen Klassen verwendet werden, was damit die Flexibilität der Anwendung erhöht.

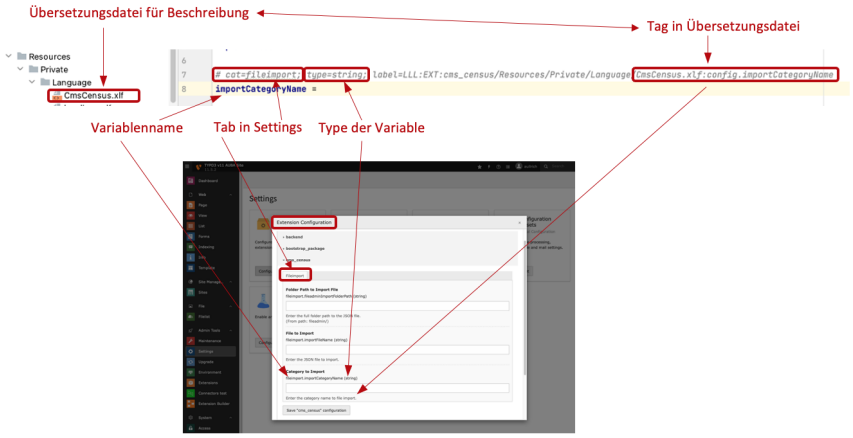

Abbildung 20: CMScensus - Extension Configuration Fileimport

So wie in der Dokumention [24] von **"External Data Import"** beschrieben, hat die Konfiguration und das Mapping auf die speziellen Datenbankfelder über die jeweilige **"TCA"**[7] des zu konfigurierenden Objektes zu erfolgen. Abbildung 21 zeigt die Konfiguration im "external" Bereich der TCA des "URL" Models. Zu sehen ist der Konfigurationsname "whatCmsJsonFileImport", der Import Connector Typ "json", die aufzurufende "uri" und die speziellen "CustomSteps", welche die eigenen Klassen zur Modifikation beinhalten. Desweitern ist die jeweilige Position der "CustomSteps" im geplanten Ablauf der späteren Importschritte zu erkennen.

Abbildung 21: CMScensus - external im TCA des URL Model

[7]Table Configuration Array - Englisch für "Tabellenkonfigurations-Array" [25]

Ein weiterer wichtiger Bereich in der TCA ist das Mapping der zu importierenden JSON Struktur auf die eigenen Datenbankfelder. Beispielhaft wird hier (siehe Abbildung 22) das Mapping für das Datenbankfeld **"name"** zu dem Feld **"url"** aus der JSON Struktur angezeigt.

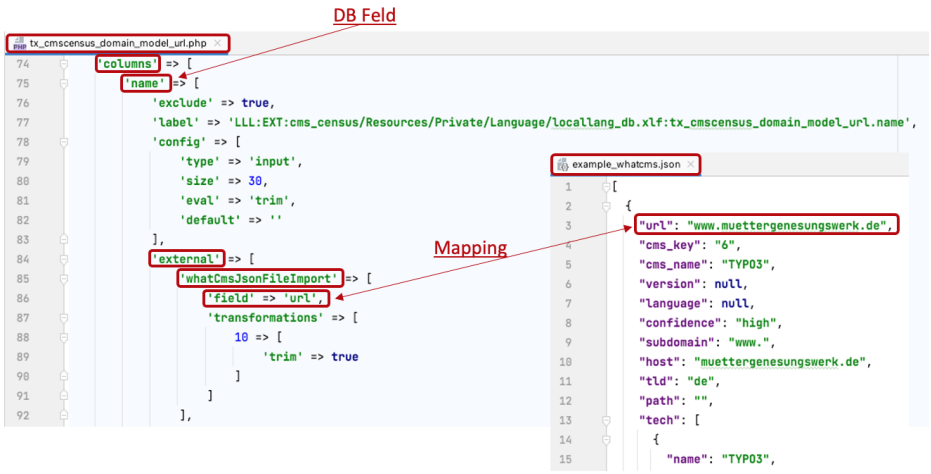

Abbildung 22: CMScensus - mapping im TCA des URL Model

Nachdem das Mapping und die Konfiguration in der TCA erfolgreich implementiert wurden, sollte im Backend unter dem Modul "Data Import" für "whatCmsJsonFileImport" nachfolgendes **"Columns Mapping"** (siehe Abbildung 23) ersichtlich sein.

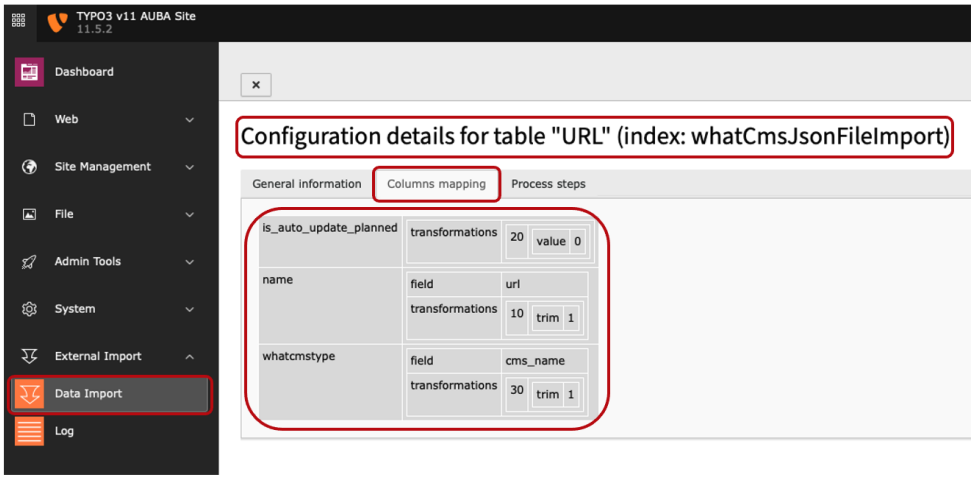

Abbildung 23: CMScensus - Data Import Columns Mapping

Unter dem Punkt "Process steps", so wie in Abbildung 24 zu erkennen, werden alle Schritte aufgeführt, welche nach dem Start des Import chronologisch abzuarbeiten sind.

Es ist dabei klar zu sehen, an welchen Punkten (1. und 9.) die eigenen CMScensus Klassen zur Modifikation des "Import Scenario" aufgerufen werden.

Abbildung 24: CMScensus - Data Import Process Steps

Im Konstruktor der Klasse "CustomizeFileImportSetupStep" (siehe Abbildung 25) von CMScensus, werden durch Verwendung des Entwurfmuster **"DI"**[8] zuerst nacheinander Objekte der Klassen "ExtensionConfiguration", "PersistenceManager" und der Klasse "CategoryRepository" erstellt. Diese Objekte werden benötigt, um auf die Konfigurationsfelder der CMScensus Extension zuzugreifen, die Datensätze des "Category Model" zu bearbeiten und entsprechende Daten persistieren [27] zu können.

Abbildung 25: CMScensus - CustomFileImportSetupStep Konstruktor

[8]Dependency Injection - Englisch für "Abhängigkeitsinjektion" [26]

Da die Klasse "CustomizeFileImportSetupStep" wie in Abbildung 26 ersichtlich von der Klasse "AbstractStep" erbt, stehen ihr alle Eigenschaften von "Importer" zur Verfügung. Sie muss die Funktion **"run()"** implementieren, da diese automatisch aufgerufen wird.

```
14  /**
15   * Class for adapting the connector URI parameter.
16   *
17   * @package AUBA\CmsCensus\Step
18   */
19  class CustomizeFileImportSetupStep extends AbstractStep
20  {
```

Abbildung 26: CMScensus - CustomFileImportSetupStep Extends

Durch den Aufruf von "run()", wird als nächstes die Funktion "setUriParameter()" aufgerufen. Diese holt sich wie in Abbildung 27 zu erkennen, im ersten Schritt den Pfad zum "fileadmin" Ordner und speichert ihn in der Variablen "$destinationAbsolutePath" ab. Im nächsten Schritt wird das "GeneralConfiguration" Array des Importer geholt und der Variablen "$generalConfiguration" zugewiesen, um aus dieser das "Parameters" Array zu extrahieren und in der Variablen "parameters" für die weitere Bearbeitung zu speichern. Im Anschluß wird das Parameters Array in einer "foreach" Schleife durchlaufen und beim Key "uri" der default Eintrag für den Pfad und die Datei durch die in den Settings der "extensionConfiguration" angegeben Werte ersetzt. Im letzten Schritt werden diese neuen Werte den "Parameters" der "GeneralConfiguration" des Importer zugewiesen und damit die alten Werte überschrieben.

CustomizeFileImportSetupStep.php

```
73      protected function setUriParameter(): void
74      {
75          $destinationAbsolutePath = GeneralUtility::getFileAbsFileName($GLOBALS['TYPO3_CONF_VARS']['BE']['fileadminDir']);
76          $generalConfiguration = $this->getImporter()->getExternalConfiguration()->getGeneralConfiguration();
77          $parameters = $generalConfiguration['parameters'];
78
79          foreach ($parameters as $key => $value) {
80              // Update the "whatCmsJsonFilePath" string from the "uri" parameter, if it exists
81              if ($key === 'uri' && strpos($value, needle: 'fileadminImportFolderPath') !== false) {
82                  $parameters[$key] = str_replace( search: 'fileadminImportFolderPath',
83                      replace: $destinationAbsolutePath . $this->extensionConfiguration['fileadminImportFolderPath'], $value);
84                  $parameters[$key] = str_replace( search: 'importFileName', $this->extensionConfiguration['importFileName'],
85                      $parameters[$key]);
86                  $parameters[$key] = str_replace( search: '//', replace: '/', $parameters[$key]);
87              }
88          }
89
90          $generalConfiguration['parameters'] = $parameters;
91          $this->getImporter()->getExternalConfiguration()->setGeneralConfiguration($generalConfiguration);
92      }
```

Abbildung 27: CMScensus - CustomFileImportSetupStep setUriParameter()

Nachdem die "setUriParameter()" Funktion erfolgreich abgearbeitet wurde, wird als nächstes die "setCategory()" Funktion aufgerufen. Diese holt sich zu Beginn, wie in Abbildung 28 ersichtlich, die "ColumnConfiguration" des Importer und speichert sie in der Variable "$columnConfiguration" ab. Anschließend wird überprüft, ob der Wert in der "extensionConfiguration" für den "importCategoryName" nicht leer ist. Ist dies der Fall, so wird im nächsten Schritt geprüft, ob die Kategorie schon in der Datenbanktabelle des "Category" Model existiert, um sie bei nicht Existenz an dieser Stelle neu anzulegen.

Abbildung 28: CMScensus - CustomFileImportSetupStep setCategory()

Im letzten Schritt der "setCategory()" Funktion wird ein neues "Transformation" Array erzeugt, welchem die entsprechende "uid" der Kategorie in einem eigenen "Key-Value" Pair[9] Array zugewiesen wird. Dieses neu erzeugte Array wird anschließend in dem "categories" Eintrag der "ColumnConfiguration" des Importer gespeichert, um die Verbindung des importierten Wertes zur festgelegten Kategorie zu sichern.

Ist der "importCategoryName" leer, so wird die gesamte Logik zur Erzeugung der Kategorieverbindung übersprungen und einfach mit dem nächsten Schritt der Importfunktion weitergemacht.

[9]Englisch für "Paar"

Die nächsten Schritte im Ablauf sind die Punkte 2 bis 8 (siehe Abb. 24 / Seite 25), welche zum Standard Szenario der "External Data Import" Extension gehören. Im anschließenden Schritt 9 wird die CMScensus Klasse **"CustomizeStoragePIDStep"** aufgerufen, welche durch Vererbung ebenfalls vom Typ "AbstractStep" ist und für die Abspeicherung der in der Konfiguration festgelegten "pid" genutzt wird.

In Abbildung 29 ist zu erkennen, daß durch den Aufruf der "run()" Funktion, lediglich die "setStoragePID()" Funktion aufgerufen wird. Diese schaut nach ob ein Eintrag für die "storagePID" in der "extensionConfiguration" vorhanden ist. Ist dies der Fall, so wird die Default "pid" der "GeneralConfiguration" des Importer durch diesen Wert überschrieben und neu gesetzt.

CustomizeStoragePIDStep.php

```
35      /**
36       * Performs the actual tasks of the step.
37       *
38       * @return void
39       * @throws \Doctrine\DBAL\Driver\Exception
40       */
41      public function run(): void
42      {
43          $this->setStoragePID();
44      }
45
46      /**
47       * Set pid to StoragePID if StoragePID is not 0.
48       *
49       * @return void
50       */
51      protected function setStoragePID(): void
52      {
53          $generalConfiguration = $this->getImporter()->getExternalConfiguration()->getGeneralConfiguration();
54          if((int)$this->extensionConfiguration['storagePID'] != 0){
55              $generalConfiguration['pid'] = (int)$this->extensionConfiguration['storagePID'];
56              $this->getImporter()->getExternalConfiguration()->setGeneralConfiguration($generalConfiguration);
57          }
58      }
59  }
```

Abbildung 29: CMScensus - CustomizeStoragePIDStep setStoragePID()

Möchte man den Import testen, so hat man die Möglichkeit, dies über das Backend Modul "Data Import" für "[whatCmsJsonFileImport]" durch drücken der "Synchonize" Taste (siehe Abb. 30) entsprechend zu starten.

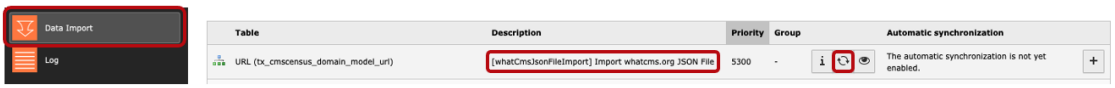

Abbildung 30: CMScensus - Data Import Synchronize

5.5.3 Erstellung Import über API Schnittstelle

Da als Grundlage für den Import die Erweiterung der "External Data Import" Extension vorgenommen wird, entspricht der Import über die "API Schnittstelle" zum großen Teil der des "Datei Import" Szenarios, welches im vorangegangenen Kapitel beschrieben wurde. Aus diesem Grund wird auf die ausführliche Erklärung der bereits beschriebenen Funktionalitäten verzichtet und nachfolgend lediglich auf die wichtigsten Unterschiede eingegangen.

Während das Mapping in der "TCA" dem gleichen Muster wie beim "Datei Import" entspricht, weicht der Konfigurationsteil im "external" Bereich etwas ab. Es ist deutlich zu erkennen (siehe Abbildung 31), daß die "uri" in der "parameters" Sektion dem zuvor in **Abschnitt 2.3.1** (Seite 4) beschriebenen **"API Request URL"** Muster folgt. Die darin eingefügten Platzhalter werden später in der "setUriParameter()" Funktion der "CustomizeSetupStep" Klasse überschrieben. Desweitern wird ersichtlich, daß es vier "CustomSteps" gibt, wobei "CustomizeStoragePIDStep" derselbe Schritt wie beim Datei Import ist und auch hier zur Speicherung der "pid" dient, so wie bereits im vorigen Abschnitt (Seite 28) ausführlich beschrieben.

tx_cmscensus_domain_model_url.php

```
39          'whatCmsApiImport' => [
40              'connector' => 'json',
41              'parameters' => [
42                  'uri' => 'https://whatcms.org/API/CMS?key=whatCmsApiKey&url=requestUrl'
43              ],
44              'data' => 'array',
45              'arrayPath' => '',
46              'referenceUid' => 'name',
47              'disabledOperations' => 'delete',
48              'customSteps' => [
49                  [
50                      'class' => AUBA\CmsCensus\Step\PlanningAutoUpdateStep::class,
51                      'position' => 'before:' . \Cobweb\ExternalImport\Step\CheckPermissionsStep::class
52                  ],
53                  [
54                      'class' => AUBA\CmsCensus\Step\CustomizeSetupStep::class,
55                      'position' => 'after:' . \Cobweb\ExternalImport\Step\CheckPermissionsStep::class
56                  ],
57                  [
58                      'class' => AUBA\CmsCensus\Step\CustomizeReferenceUIDMappingStep::class,
59                      'position' => 'before:' . \Cobweb\ExternalImport\Step\ValidateDataStep::class
60                  ],
61                  [
62                      'class' => AUBA\CmsCensus\Step\CustomizeStoragePIDStep::class,
63                      'position' => 'before:' . \Cobweb\ExternalImport\Step\StoreDataStep::class
64                  ]
65              ],
66              'priority' => 5310,
67              'description' => 'Import CMS Type for URL from whatcms.org'
68          ]
69      ]
70  ],
```

Abbildung 31: CMScensus - external API im TCA des URL Model

Da man über die API von WhatCMS.org nur einen Request aller 10 Sekunden stellen darf, musste für CMScensus eine spezielle Lösung erarbeitet werden. Im Lösungsansatz wurde die Idee der Planung von Updates nach dem Muster des "Bubblesort" [28] Algorithmus gefunden. Dabei sollen im ersten Schritt die URLs, für welche das jeweilige CMS bei WhatCMS.org ermittelt werden soll, mit einem **"is_auto_update_planned"** Flag und dem Wert **"TRUE"** versehen und anschließend bei jedem Aufruf des Importskript wie die Blasen beim Bubblesort Algorithmus einer nach dem anderen abgearbeitet werden. Nach erfolgreicher Abarbeitung eines URL Updates wird für diese URL das "is_auto_update_planned" Flag auf **"FALSE"** gesetzt und damit beim nächsten Import die nächste URL verwendet.

Schaut man sich in Abbildung 32 die Funktionsweise der "PlanningAutoUpdateStep" Klasse genauer an, so ist zu erkennen, daß im ersten Schritt in der "run()" Funktion geprüft wird, ob überhaupt eine URL mit dem "is_auto_update_planned" Flagwert "TRUE" in der Datenbank existiert. Ist dies der Fall, so wird **keine** Planung durchgeführt und direkt mit dem nächsten Schritt des Importablaufes fortgefahren.
Ist dies hingegen **nicht** der Fall, so wird die Funktion "planningAutoUpdate()" aufgerufen, welche alle URLs in der Datenbank einzeln durchläuft und prüft, ob die Updates nur einmalig (only_next_auto_update == 1) oder immer (every_auto_update == 1) erfolgen sollen und ob es sich bei dem Eintrag um keinen Vorschlag handelt (is_proposal == 0). Ist dies für die jeweilige URL der Fall, so wird für diese der "is_auto_update_planned" Flagwert auf 1 (TRUE) gesetzt. Um alle nur einmal geplanten Updates für die nächste Planung zu deaktivieren, wird der Wert für "only_next_auto_update" pauschal mit 0 (FALSE) versehen.

Abbildung 32: CMScensus - API Import PlanningAutoUpdateStep

Da die "External Data Import" Extension durch die CMScensus Extension modifiziert wurde, kann sie den Datensatz der jeweiligen URL nicht automatisch zuordnen. Aus diesem Grund wird durch die **"CustomizeReferenceUIDMappingStep"** Klasse das entsprechende Mapping für das Feld **"name"** zum jeweiligen URL Eintrag vorgenommen.

Zum Zeitpunkt des Aufruf der "CustomizeReferenceUIDMappingStep" Klasse, welche ebenfalls von "AbstractStep" erbt, stehen die bereits bei WhatCMS.org abgefragten Daten zur URL im **"RawData"** Array des Importer zur Verfügung. Durch den Aufruf der "run()" Funktion, wird die "addRequestUrlToRawData()" Funktion ausgeführt. Im ersten Schritt wird das "RawData" Array des Importer (siehe Abbildung 33) in der Variablen "$rawData" gespeichert. Diesem wird anschließend durch ein "Key-Value" Pair[10] Array die URL Adresse als "requestURL" Eintrag hinzugefügt. Konnte WhatCMS.org kein passendes CMS für die abgefragte URL finden, so ist das "Records" Array des Importer leer, was im "if() - Zweig" der "addRequestUrlToRawData()" Funktion überprüft wird. Wie in der Abbildung 33 zu erkennen ist, wird im Falle eines gefüllten "Records" Array diesem ebenfalls ein "Key-Value" Pair Array mit der URL Adresse als "name" hinzugefügt. Ist das "Records" Array hingegen leer, so wird ein Default Eintrag mit dem Inhalt **"CMS Not Detected"** für den **"whatcmstype"** gesetzt. Desweiteren werden die Einträge für "is_auto_update_planned" und "every_auto_update" auf 0 (FALSE) gesetzt, damit diese URL aus der zukünftigen, weiteren Updateplanung entsprechend ausgeschlossen wird.

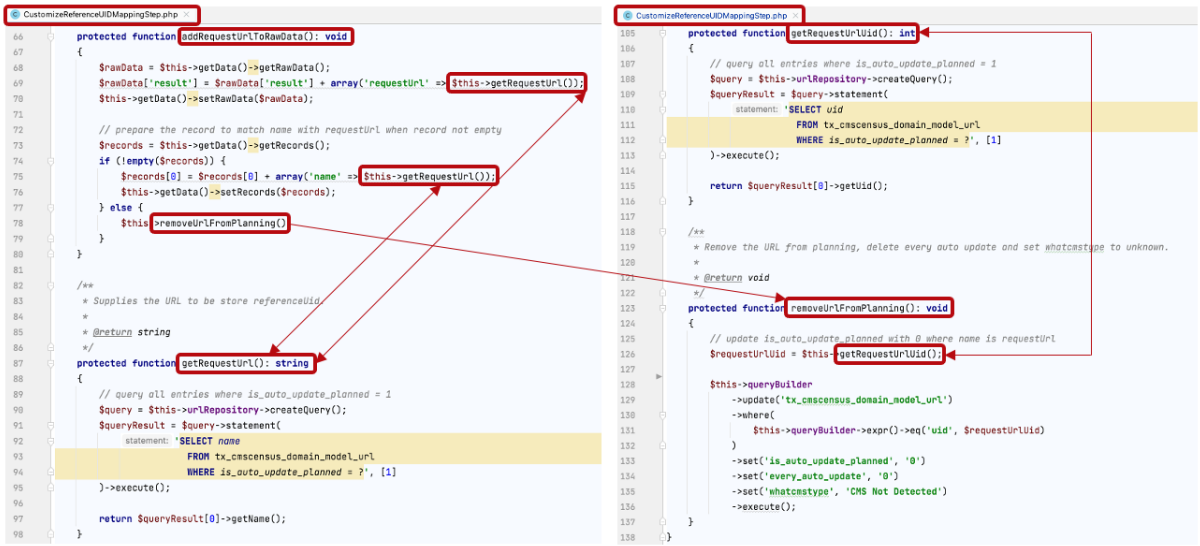

Abbildung 33: CMScensus - API Import CustomizeReferenceUIDMappingStep

[10]Englisch für "Paar"

5.5.4 Plugins für Content verfügbar machen

In der CMScensus Extension existieren zwei Plugins, welche im Contentbereich auswählbar sein sollen. Es handelt sich zum einen um das Plugin **"ProposalCmsCensus"**, welches für den Prozess neuer Vorschläge verantwortlich ist und zum anderen um das für die Chartdarstellung verantwortliche **"ChartCmsCensus"** Plugin, welches sich um die Darstellung der ausgewerteten Daten einer zuvor ausgewählten Kategorie kümmert.

Wie in Abbildung 34 zu erkennen ist, wird die Registrierung der Plugins in der "tt_content" PHP Datei durchgeführt. Neben den beiden Plugins wird noch eine Flexform [29] registriert, welche die Möglichkeit der Auswahl einer Kategorie im "cmscensus_chartcmscensus" Plugin hinzufügt. Die Flexform selbst bezieht ihre Struktur dabei aus einer "XML" [30] Datei.

Abbildung 34: CMScensus - Content Plugin Registration

Nachdem der Setup in der "tt_content.php" Datei erfolgreich durchgeführt wurde, sollten im "Create new content element" Fenster des Backend unter dem Tab "Plugins" die beiden CMScensus Plugins, so wie in Abbildung 35 ersichtlich, angezeigt werden und entsprechend als **"Content Element"** auswählbar sein.

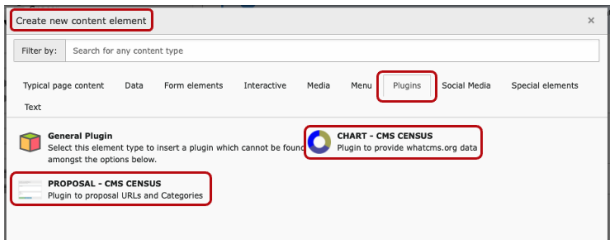

Abbildung 35: Content Element Plugins

Damit nach dem Einfügen des jeweils ausgewählten Plugins im Content auch der richtige Controller mit der jeweiligen "Action-Funktion" ausgeführt wird, muss in der Datei "ext_localconf.php" die entsprechende Konfiguration durgeführt werden.

Es wird wie in Abbildung 36 ersichtlich, der **"configurePlugin()"** Funktion, sowohl der Name der Extension, als auch der Name des Plugin, für welches die Konfiguration zu erfolgen hat übergeben. Desweiteren werden die Controller des Plugin, sowie ihre aufrufbaren "Action-Funktionen" TYPO3 bekannt gemacht. Die erste Action Funktion wird dabei als **Default** für den jeweiligen Controller definiert und beim Aufruf ohne Action ausgeführt. In Abbildung 36 ist beispielhaft zu erkennen, daß für die Default Action des "ChartController" die Action "show" angegeben wird, welche als Funktion in der "ChartController" Klasse mit der Bezeichnung "showAction()" existieren muss.

ext_localconf.php

```php
(static function() {
    \TYPO3\CMS\Extbase\Utility\ExtensionUtility::configurePlugin(
        'CmsCensus',
        'Chartcmscensus',
        [
            ChartController::class => 'show',
            AjaxController::class => 'cmsPerCategoryUrls'
        ],
        // non-cacheable actions
        [
            ChartController::class => 'show',
            AjaxController::class => 'cmsPerCategoryUrls'
        ]
    );

    \TYPO3\CMS\Extbase\Utility\ExtensionUtility::configurePlugin(
        'CmsCensus',
        'Proposalcmscensus',
        [
            ProposalController::class => 'addUrlForm, addCategoryForm, createUrl, createCategory'
        ],
        // non-cacheable actions
        [
            ProposalController::class => 'addUrlForm, addCategoryForm, createUrl, createCategory'
        ]
    );
```

Abbildung 36: CMScensus - Configure Plugin

Es ist möglich, weitere **Konfigurationseinstellungen** in der "ext_localconf.php" Datei vorzunehmen, um die Verhaltensweise der Plugins zu steuern. Für "CMScensus" wurden zum Beispiel noch weitere Einstellungen für die anzuzeigenden **Icons**, die **Titel** und die **Beschreibungen** vorgenommen.

5.5.5 Erstellung Proposal Funktionalität

Wie im **Abschnitt 5.5.4** (Seite 33 / Abbildung 36) in der abgebildeten Konfiguration zu erkennen ist, wird als Defaultfunktion des "ProposalController" beim Aufruf einer Seite, welche im Inhalt das Content Plugin **"proposalcmscensus"** verwendet, als erstes die "addUrlFormAction()" Funktion ausgeführt.

In der "addUrlFormAction()" Funktion wird wie in Abbildung 37 ersichtlich im ersten Schritt geprüft, ob das "enableProposalWithAuth" Flag im "extensionConfiguration" Array auf **"TRUE"** (1) gesetzt ist und wenn ja, ob es sich bei dem Besucher der Seite um einen angemeldeten Frontendbenutzer handelt. Ist das Flag auf 1 und der Besucher nicht angemeldet, so wird er zu der Seite mit der im "extensionConfiguration['loginPID']" Array angegebenen "pid" durch ein ausgelöstes "redirect()" weitergeleitet. Diese Seite sollte den Benutzer entsprechend zur Anmeldung auffordern.

ProposalController.php

```
85      /**
86       * Show form to add an new URL Proposal
87       *
88       * @param Url $url
89       */
90      public function addUrlFormAction(Url $url = null): void
91      {
92          // Checking whether the offer with authentication is activated
93          // and forwarding if the user is not logged in
94          if($this->extensionConfiguration['enableProposalWithAuth'] == 1){
95              $context = GeneralUtility::makeInstance(Context::class);
96              if (!$context->getPropertyFromAspect('frontend.user', 'isLoggedIn')) {
97                  $this->redirect( actionName: null,  controllerName: null,  extensionName: null,  arguments: null,
98                      $this->extensionConfiguration['loginPID']);
99              }
100         }
101
102         $this->view->assign('url', $url);
103         $this->view->assign('categories', $this->categoryRepository->findAll());
104     }
```

Abbildung 37: CMScensus - ProposalController addUrlFormAction()

Da der "ProposalController" von der Klasse "ActionController" erbt, steht ihm die Funktionalität eines View Objektes, welches zur Verarbeitung der darzustellenden Daten verwendet wird, zur Verfügung. Diesem wird durch Benutzung der "view()" Funktion unter dem Schlüssel "url" das URL Objekt und unter dem Schlüssel "categories" alle Kategorieeinträge der Rückgabe der "findAll()" Funktion des "categoryRepository" übergeben (siehe Abbildung 37).

Bevor es zur Browserausgabe kommt wird, wie in Abbildung 38 zu erkennen ist, das Fluid [31] Styled Template **"AddUrlForm.html"** geladen, welches dem Namen der zuvor ausgeführten Controller Action "addUrlForm" entspricht. In dem Template wird wiederum das Layout "Default.html" geöffnet und in diesem die Section "content" des "AddUrlForm.html" Template gerendert. Beim Prozess des Rendering wird das angegebene Partial "UrlForm.html" eingebunden und alle darin enthaltenden Variablen mit den dazugehörigen Werten befüllt. Nach erfolgreichem Abschluß des kompletten Vorgangs wird das in Abbildung 38 ersichtliche Formular im Browser ausgegeben.

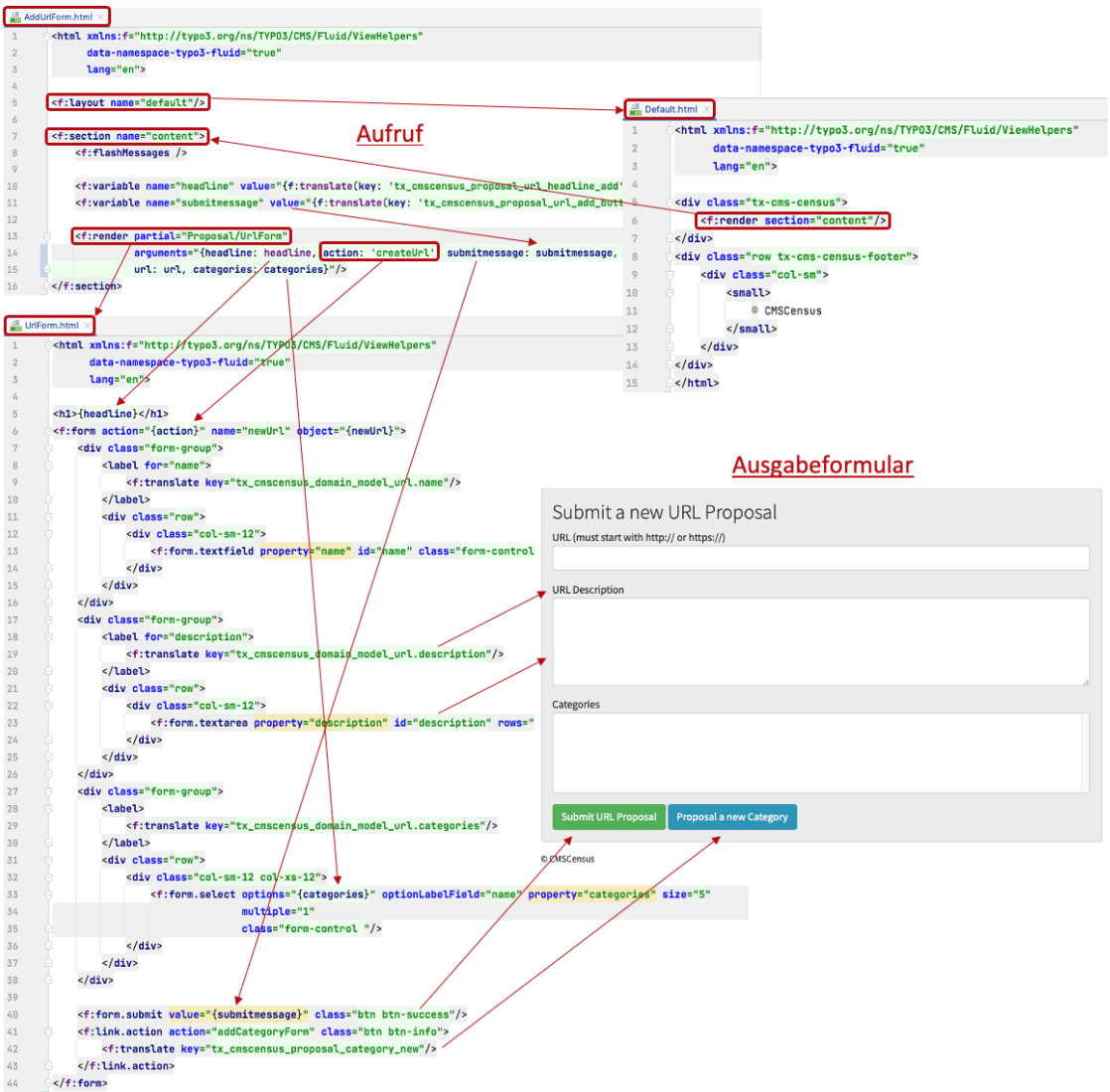

Abbildung 38: CMScensus - Fluid Template Proposal

Nach ausfüllen des Formulars und anschließender Betätigung des zum absenden vorgesehenen Button **"Submit URL Proposal"**, wird die Action **"createUrl"** an den Proposal Controller gesandt, wo sie die direkte Ausführung der "createUrlAction()" Funktion zur Folge hat.

Durch Aufruf der **"createUrlAction()"** Funktion (siehe Abbildung 39), wird im ersten Schritt der Wert des Namen von dem an die Funktion übergebenen "URL" Objekt auf Korrektheit geprüft. Dies geschieht durch Verwendung der von PHP zur Verfügung stehenden "filter_var()" [32] Funktion. Ist der Name der URL korrekt, so wird das in dem übergebenen URL Objekt vorhandene "is_proposal" Flag auf "TRUE" gesetzt und anschließend das komplette URL Objekt dem "urlRepository" hinzugefügt. Dieses hinzufügen hat die automatische Speicherung des URL Objektes in der Datenbank zur Folge, um welche sich das "UrlRepository" selbständig kümmert. Im letzten Schritt wird die Controller eigene Funktion "finishCreation()" aufgerufen, welche die Weiterleitung zur "addUrlFormAction()" Funktion zur Folge hat.

ProposalController.php

```
110     public function createUrlAction(\AUBA\CmsCensus\Domain\Model\Url $newUrl)
111     {
112         if (filter_var($newUrl->getName(), filter: FILTER_VALIDATE_URL)) {
113             $newUrl->setIsProposal(true);
114             $this->urlRepository->add($newUrl);
115             $this->finishCreation( redirectDestination: 'addUrlForm');
116         } else {
117             $errorMessageBody = LocalizationUtility::translate(
118                 'tx_cmscensus_flashmessage_proposal-error.url',
119                 'CmsCensus'
120             );
121             $this->addFlashMessage( messageBody: $errorMessageBody . ' => [' . $newUrl->getName() . ']', messageTitle: '',
122                  severity: AbstractMessage::ERROR);
123             $this->redirect( actionName: 'addUrlForm');
124         }
125     }
```

Abbildung 39: CMScensus - ProposalController createUrlAction()

Ist die Validierung der URL in der "createUrlAction()" Funktion fehlgeschlagen, so wird eine eigene Fehlermeldung erzeugt und dem "FlashMessage" Stack des Controllers hinzugefügt (siehe Abbildung 39). Anschließend wird durch Aufruf der "redirect()" Funktion zur "addUrlFormAction()" Funktion weitergeleitet. Der zuvor mit der Fehlermeldung gefüllte "FlashMessage" Stack wird im View des "AddFormUrl.html" Template an der Stelle "flashMessages" (siehe Abbildung 38 / Seite 35) eingesetzt und dadurch als Hinweis im gerenderten Formular des Browser ausgegeben.

Der Vorschlagprozess für die Kategorien ist technisch sehr ähnlich dem der bereits ausführlich beschriebenen Funktionsweise der URLs. Aus diesem Grund wird auf die weitere genaue Beschreibung im Zuge der vorliegenden Arbeit verzichtet.

5.5.6 Erstellung Chart Funktionalität

Auch für die Chartausgabe ist wie im **Abschnitt 5.5.4** (Seite 33 / Abbildung 36) in der abgebildeten Konfiguration zu erkennen, das als Defaultfunktion beim Aufruf einer Seite des "ChartController", welche im Inhalt das Content Plugin **"chartcmscensus"** verwendet, als erstes die "showAction()" Funktion ausgeführt wird. Es ist außerdem zu sehen, das als Defaultfunktion für den "AjaxController" beim Aufruf die Funktion "cmsPerCategoryUrlsAction()" entsprechend Anwendung findet.

Schaut man sich die beiden Controller genauer an (siehe Abbildung 40), so ist zu erkennen, das beide Defaultfunktionen zu Beginn versuchen die "uid" der Kategorie zu ermitteln und entsprechend in der Variablen "$categoryUid" zu speichern. Im nächsten Schritt wird via **"DI"**[11] ein Objekt der "CategoryUrls" Klasse erzeugt und in der Klassenvariablen "categoryUrls" gespeichert. Danach wird die "countCmsOfCategoryUrls()" Funktion des "CategoryUrls" Objekt mit der "categoryUid" als Funktionsparameter aufgerufen. Ihr Rückgabewert ist ein Array mit den durch die Funktion berechneten statistischen Daten zur zuvor übegebenen "categoryUid". Die im Array gespeicherten Daten werden in der Defaultfunktion des "ChartController" dem View Objekt zur Ausgabe mit verschiedenen "Key-Value" Pairs[12] zugewiesen. In der Funktion des "AjaxController" werden die Werte des Array erst in ein JSON Objekt gepackt und anschließend dem Aufrufer der Funktion zurückgegeben.

Abbildung 40: CMScensus - ChartController und AjaxController

Die Hauptaufgabe der "countCmsOfCategoryUrls()" Funktion des "CategoryUrls" Objekt liegt darin, die Anzahl der verwendeten "CMS" pro Kategorie prozentual ins Verhältnis zu setzen und als entsprechende Werte zurückzugeben. Desweitern wird im Rückgabearray die Liste der verwendeten Farben und CMS Namen gespeichert. Somit können die Daten an den unterschiedlichsten Stellen individuell ausgegeben oder weiterverarbeitet werden.

[11] Dependency Injection - Englisch für "Abhängigkeitsinjektion" [26]
[12] Englisch für "Paare"

Genau wie beim "Proposal" Plugin (siehe Abschnitt 5.5.5 / Seite 35), wird auch beim "Chart" Plugin vor der Browserausgabe ein Fluid [31] Styled Template geladen. Dieses entspricht im Namen wieder der zuvor ausgeführten Action. Da beim "ChartController" die Defaultaction **"show"** lautet, wird entsprechend das **"Show.html"** Template geladen.

Wie in Abbildung 41 zu erkennen ist, wird im ersten Schritt des "Show.html" Template das "Default.html" Layout eingebunden, welches wiederum die Sektion "content" des "Show.html" Template zum Rendering ausführt. In diesem wird danach die Script Konstante "ajaxUrl" erstellt und mit der für die Anfrage des "AjaxController" zuständigen "URL" befüllt, um sie in der dazugehörigen **"cmscensus.js"** JavaScript Datei für den nachfolgenden "AjaxRequest" zur Erstellung der Chartdarstellung zu verwenden. Im nächsten Schritt wird die Überschrift im Bereich der Codezeilen 15 bis 17 durch die in der "locallang.xlf" Übersetzungsdatei für den Schlüssel "tx_cmscensus_chart_url_headline" passende Übersetzung hinzugefügt.

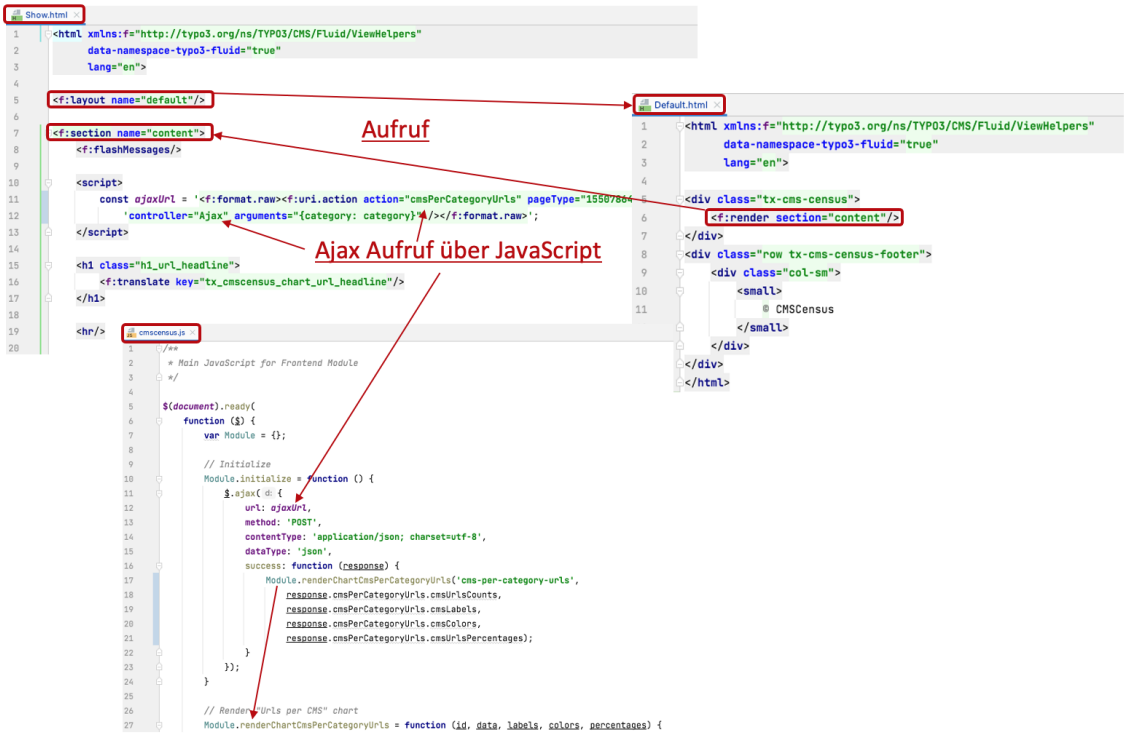

Abbildung 41: CMScensus - Fluid Template Show und Ajax

In den Codezeilen 20 bis 45 wird der Inhalt des Kopfbereiches für die Ausgabe des "Chart" Plugin mit den zusammenfassenden Werten für die ausgewählte "Kategorie", sowie die Gesamtzahl aller betrachteten "URLs" erzeugt.

Im letzten Teil des "Show.html" Template wird, wie in Abbildung 42 erkennbar ist, zuerst die Tabelle mit den statistischen Daten erzeugt, welche die Verteilung der "CMS" aller "URLs" ausgibt. Dazu wird das Datensatz Array "tableCmsCountOfCategoryUrls" in einer "For-Schleife" durchlaufen und jeder Wert Zeile für Zeile der Tabelle hinzugefügt. Die letzte Zeile der Tabelle beinhaltet die Zusammenfassung aller Teilwerte, um sie als "Total" auszugeben.

Das in der Zeile 85 eingefügte "canvas" Tag wird benötigt, um die Ausgabe des Chart zu erzeugen. Durch die angegebene ID "cms-per-category-urls" wird eine Verbindung zwischen dem **"Canvas"**[13] Objekt und der JavaScript Datei **"cmscensus.js"** hergestellt, um anschließend die statistischen Daten des ausgeführten JavaScript in Chartform zur Anzeige zu bringen.

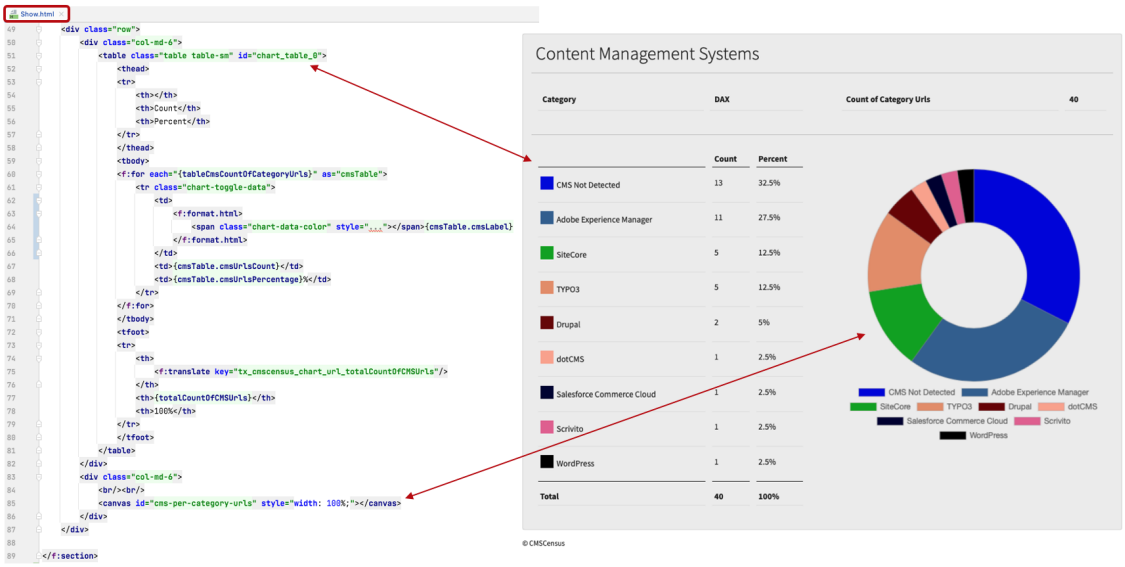

Abbildung 42: CMScensus - Fluid Template Show und Chart Darstellung

Da die Ausführung des "Ajax-Request" in der JavaScript Datei "cmscensus.js" erst bei die Browserausgabe des "Show.html" Template angeschoben wird, ist die Anzeige der Chartform im Gegensatz zur Ausgabe der Datentabelle je nach Antwortzeit des Servers verzögert zu sehen.

[13]Englisch für "Leinwand oder Gemälde"

6 Fazit und Ausblick

Im Folgenden wird der Entwicklungsprozess von der Idee, über die Konzeption, bis hin zur Implementierung des Prototyp der TYPO3 Extension "CMScensus" kritisch betrachtet und Erfahrungen, sowie Erkenntnisse, welche während des Prozesses auftraten vorgestellt und beschrieben. Im Anschluß werden Vorschläge und Anregungen zur weiteren Optimierung oder Weiterentwicklung des Prototyps aufgezeigt.

6.1 Fazit

Das Ziel der Arbeit war die Entwicklung einer Extension für TYPO3 (Version 11), welche erstellt und anschließend installier-, sowie ausführbar sein muss. Der entwickelte Prototyp soll für den Onlinebetrieb einsatzbereit und für die Öffentlichkeit im Internet erreichbar sein. Desweiteren sollte die Extension die Möglichkeit zur Eingabe neuer Kategorien und URL's, sowie die visuelle Auswertung der statistischen Daten über die Verteilung von benutzten CMS einer Kategorie in Tabellen- und Chartform für Frontendbenutzer zur Verfügung stellen. Dieses Ziel konnte im Rahmen der vorliegenden Arbeit erfolgreich umgesetzt werden. Der Prototyp der Extension läuft aktuell[14] auf einem Server mit installiertem TYPO3 (Version 11), welcher öffentlich im Internet unter der Domain **"www.cmscensus.eu"** erreichbar ist.

Der gesamte Entwicklungsprozess folgte einer im Vorfeld selbst definierten und festgelegten Struktur. Während zu Beginn erst das Verständnis für die Datenquelle, sowie für die Funktionsweise der API Schnittstelle von "WhatCMS.org" aufgebaut werden musste, folgten im Anschluß eigene Tests zur Implementierung und Weiterverarbeitung. Dabei wurde die Erkenntnis gewonnen, daß es für die Umsetzung der Importfunktionalität besser ist, die bereits bestehende TYPO3 Extension "External Data Import" zu verwenden und diese anschließend für die eigenen Zwecke zu erweitern. Auch die Umsetzung des "Domain-Driven Design" Pattern erforderte im Vorfeld die Wissenserweiterung durch verschiedene Quellen (Bücher und Internet), damit es im Anschluß Schritt für Schritt nach einem festen Muster abgearbeitet werden konnte.

Für die Entwicklung wurde zur Versionierung das freie Open Source Kontrollsystem Git [14] verwendet und das dazugehörige "Git Repository" auf "GitHub" eingerichtet. Unter der URL **"https://github.com/usadmin77/bachelorarbeit.git"** ist für ausgewählte Benutzer das Repository zugänglich und somit der komplette Entwicklungszyklus durch die automatisch geführten Logeinträge jederzeit lückenlos nachvollziehbar. Im Zuge der Entwicklung konnte die Erfahrung gemacht werden, daß die Funktionalität der Wiederherstellung eines zuvor gespeicherten Projektzustandes aus der Version im "Git Repository" absolute Flexibilität und Sicherheit mit sich bringt und damit die Qualität der Entwicklung deutlich verbessert.

[14] Stand Dezember 2021

Die funktionalen und technischen Anforderungen an die Extension konnten im Ergebnis erfolgreich umgesetzt und implementiert werden. Im Laufe der Entwicklung kamen neue Erkenntnisse für besseren Programmierstil hinzu. So wurden zum Beispiel zu Beginn Datenbankabfragen, welche das Model betreffen, via "QueryBuilder" durchgeführt. Auf diese Art der Abfrage sollte allerdings gemäß der TYPO3 Dokumentation [33] verzichtet werden und stattdessen die Abfrage über ein "Repository" erfolgen.

6.2 Ausblick

Der aktuelle Stand des Prototyps der TYPO3 Extension "CMScensus" bietet eine gute Ausgangslage für eine entsprechende Weiterentwicklung. So empfiehlt es sich zum Beispiel beim "Proposal" Plugin die Überprüfung des angemeldeten Frontendbenutzer auf den Vorschlag der Kategorie zu erweitern. Da die Klassen der "CustomSteps" zu Anfang mit dem "QueryBuilder" entwickelt wurden und dies bewusst beispielhaft im Code an einigen Stellen erhalten blieb, sollten diese durch die "Repositoy" Abfrage ersetzt werden.

Für die bessere Bedienung empfiehlt es sich die Extension um weitere "Backend" Funktionalität zu erweitern. So könnte zum Beispiel eine getrennte Ansicht aller URLs einer Kategorie erzeugt werden. Auch eine Filterung nach verschiedensten Gesichtspunkten könnte die Übersichtlichkeit für Redakteure verbessern. Die Importfunktionalität kann durch ein Upload Formular mit anschließender Ausführung des Import erweitert werden.

Da die "CMScensus" Extension auf jedem TYPO3 (Version 11) Server installierbar ist, besteht somit auch die Möglichkeit diese nach eigenen Anforderungen individuell anzupassen, zu erweitern oder entsprechend weiterzuentwickeln.

7 Quellen und Literaturverzeichnis

Literatur

[1] Wikipedia. *Content-Management-System – Wikipedia*. URL: https://de.wikipedia.org/wiki/Content-Management-System (besucht am 30. 11. 2021).

[2] CMS Matrix. *The CMS Matrix - cmsmatrix.org - The Content Management Comparison Tool*. URL: http://www.cmsmatrix.org (besucht am 30. 11. 2021).

[3] TYPO3.org. *Development Roadmap for TYPO3 CMS*. URL: https://typo3.org/cms/roadmap (besucht am 30. 11. 2021).

[4] WhatCMS.org. *Detect which CMS a site is using - What CMS?* URL: https://whatcms.org/ (besucht am 30. 11. 2021).

[5] WhatCMS.org. *About - What CMS?* URL: https://whatcms.org/About (besucht am 01. 12. 2021).

[6] Wikipedia. *Uniform Resource Locator – Wikipedia*. URL: https://de.wikipedia.org/wiki/Uniform_Resource_Locator (besucht am 01. 12. 2021).

[7] WhatCMS.org. *Documentation - What CMS?* URL: https://whatcms.org/Documentation (besucht am 01. 12. 2021).

[8] Wikipedia. *JavaScript Object Notation – Wikipedia*. URL: https://de.wikipedia.org/wiki/JavaScript_Object_Notation (besucht am 01. 12. 2021).

[9] Wikipedia. *Volkszählung – Wikipedia*. URL: https://de.wikipedia.org/wiki/Volksz%C3%A4hlung (besucht am 01. 12. 2021).

[10] DEV Insider. *Was bedeutet LTS?* URL: https://www.dev-insider.de/was-bedeutet-lts-a-959219 (besucht am 04. 12. 2021).

[11] The PHP Group. *PHP: PHP 7 ChangeLog*. URL: https://www.php.net/ChangeLog-7.php (besucht am 04. 12. 2021).

[12] The Composer Group. *Composer*. URL: https://getcomposer.org/changelog/2.1.14 (besucht am 04. 12. 2021).

[13] Helhum. *TYPO3-Console/TYPO3-Console: Console command for TYPO3 CMS*. URL: https://github.com/TYPO3-Console/TYPO3-Console (besucht am 04. 12. 2021).

[14] git. *Git*. URL: https://git-scm.com (besucht am 04. 12. 2021).

[15] Wikipedia. *Model View Controller – Wikipedia*. URL: https://de.wikipedia.org/wiki/Model_View_Controller (besucht am 05. 12. 2021).

[16] Localdev Foundation. *DDEV-Local Documentation*. URL: https://ddev.readthedocs.io/en/stable (besucht am 06. 12. 2021).

[17] Docker. *Empowering App Development for Developers — Docker*. URL: https://www.docker.com (besucht am 06. 12. 2021).

[18] TYPO3.org. *Installing TYPO3 — Getting Started main documentation*. URL: https://docs.typo3.org/m/typo3/tutorial-getting-started/main/en-us/Installation/Install.html#install (besucht am 06. 12. 2021).

[19] TYPO3.org. *TYPO3 Extension 'Extension Builder'*. URL: https://extensions.typo3.org/extension/extension_builder (besucht am 08. 12. 2021).

[20] Wikipedia. *Grafische Benutzeroberfläche – Wikipedia*. URL: https://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che (besucht am 08. 12. 2021).

[21] TYPO3.org. *TYPO3 Extension 'The Official TYPO3 Introduction Package'*. URL: https://extensions.typo3.org/extension/introduction (besucht am 08. 12. 2021).

[22] Wikipedia. *Redundanz (Informationstheorie) – Wikipedia*. URL: https://de.wikipedia.org/wiki/Redundanz_(Informationstheorie) (besucht am 08. 12. 2021).

[23] TYPO3.org. *TYPO3 Extension 'External Data Import'*. URL: https://extensions.typo3.org/extension/external_import (besucht am 08. 12. 2021).

[24] TYPO3.org. *General TCA configuration — External Import 6.0 documentation*. URL: https://docs.typo3.org/p/cobweb/external_import/6.0/en-us/Administration/GeneralTca/Index.html (besucht am 10. 12. 2021).

[25] TYPO3.org. *TCA Reference — TCA Reference main documentation*. URL: https://docs.typo3.org/m/typo3/reference-tca/main/en-us/Index.html (besucht am 10. 12. 2021).

[26] Wikipedia. *Dependency Injection – Wikipedia*. URL: https://de.wikipedia.org/wiki/Dependency_Injection (besucht am 12. 12. 2021).

[27] Wikipedia. *Persistenz (Informatik) – Wikipedia*. URL: https://de.wikipedia.org/wiki/Persistenz_(Informatik) (besucht am 12. 12. 2021).

[28] Wikipedia. *Bubblesort – Wikipedia*. URL: https://de.wikipedia.org/wiki/Bubblesort (besucht am 14. 12. 2021).

[29] TYPO3.org. *Flexforms — TYPO3 Explained main documentation*. URL: https://docs.typo3.org/m/typo3/reference-coreapi/main/en-us/ApiOverview/FlexForms/Index.html (besucht am 16. 12. 2021).

[30] Wikipedia. *Extensible Markup Language – Wikipedia*. URL: https://de.wikipedia.org/wiki/Extensible_Markup_Language (besucht am 14. 12. 2021).

[31] TYPO3.org. *Fluid Template Engine*. URL: https://typo3.org/fluid (besucht am 21. 12. 2021).

[32] PHP. *PHP: filter_var - Manual*. URL: https://www.php.net/manual/de/function.filter-var.php (besucht am 21. 12. 2021).

[33] TYPO3.org. *Individual database queries — Developing TYPO3 Extensions with Extbase and Fluid main documentation*. URL: https://docs.typo3.org/m/typo3/book-extbasefluid/main/en-us/6-Persistence/3-implement-individual-database-queries.html (besucht am 23. 12. 2021).

[34] J. Böhm. *Erfolgsfaktor Agilität*. Hrsg. von Springer Vieweg. 2019, S. 119 ff.

[35] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Hrsg. von Addison Wesley. 2003.

[36] Michael Schams. *TYPO3 Extbase: Moderne Extension-Entwicklung mit Extbase und Fluid*. Hrsg. von Independently published. 2010.

[37] Mark Richards. *Software Architecture Patterns*. Hrsg. von Inc. Reilly Media. URL: `https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html` (besucht am 05. 12. 2021).