Die Senator für Finanzen

Richtlinien und verpflichtende Vorgaben bei jeder Form der Neu- und Weiterentwicklung von Modulen und Funktionen innerhalb einer KoGIs-Instanz – sowohl bei Nutzung der KoGIs-Basismodule als auch der erweiterten Projektmodule („Projektmaster“)

Version vom März 2020

Freie Hansestadt Bremen

Impressum

Herausgeber
Senator für Finanzen
IT-Querschnitt und IT-Basiskomponenten – Referat 41
Rudolf Hilferding Platz 1
28195 Bremen

Kontaktadresse
Senator für Finanzen
IT-Querschnitt und IT-Basiskomponenten – Referat 41
Rudolf Hilferding Platz 1
28195 Bremen
E-Mail: it-basiskomponenten@finanzen.bremen.de
URL: www.finanzen.bremen.de

1. Präambel 4

2. Abstimmung aller Eigenentwicklungen mit dem KoGIs 5

3. Integration von Eigenentwicklungen innerhalb der bereit gestellten KoGIs-Entwicklungsinstanz 6

4. Verortung der Eigenentwicklungen innerhalb der KoGIs-Instanz 7

5. Gestaltungsmöglichkeiten bei Verwendung des Projektmasters 8

6. Sicherstellung der Übertragbarkeit 9

6.1 Einhaltung des Nummernraums für Templates 9
6.2 Einhaltung der Benamung für Templates 9
6.3 Einhaltung der Namenskonventionen für Workspaces, Rollen und Rechten 10
6.4 Vermeidung statischer Adressierung und Verwendung der GSId 10

7. Sicherstellung der Qualität und Performance 11

8. Sicherstellung der Barrierefreiheit 12

9. Weitere Empfehlungen für die Entwicklung 13

9.1 Namensräume 13

10. Konsequenzen und Ausblick auf zukünftige Weiterentwicklungen (auch der KoGIs-Basismodule) 17

11. Aktualisierungen 19

1. Präambel

Das Kompetenzzentrum für die Gestaltung der Informationssysteme (KoGIs) stellt Dienststellen, Gesellschaften, Einrichtungen, Stiftungen und Eigenbetrieben die KoGIs-Module und Erweiterungen zur Verfügung.

Für die Nutzung und vor allem die (Weiter-)Entwicklung der Module sind die im Folgenden aufgeführten Vorgaben strikt einzuhalten.

Die Dienststellen sorgen dafür, dass diese Vorgaben eingehalten und auch an die externen Agenturen weitergeleitet werden.

Bei allen Eigenentwicklungen sind auch die im Dokument Gestaltungsmöglichkeiten und -vorgaben bei Eigenentwicklungen aufgeführten Punkte einzuhalten.

2. Abstimmung aller Eigenentwicklungen mit dem KoGIs

Sämtliche Weiterentwicklungen von Funktionalitäten sowie Neuentwicklungen auf den KoGIS-Instanzen sind zunächst bereits **in der Planungsphase dem KoGIs zu melden**, damit sichergestellt werden kann, dass keine Parallelentwicklungen stattfinden.

Bei der Konzepterstellung und **vor Beginn der Umsetzung sind die Entwicklungen mit dem KoGIs abzustimmen und zu koordinieren**.

Dazu genügt eine kurze Darstellung der geplanten Anwendung, aus der hervorgeht:

- worum es sich bei der Anwendung grob handelt (Grobkonzept, Anforderungsdefinition, Pflichtenheft),
- ob Daten aus anderen Systemen in die KoGIs-Instanz integriert werden (Import, Replikation, Transfer),
- ob Daten in andere Systeme übertragen bzw. überführt werden (Export, Replikation, Transfer),
- in welchen Zyklen Im- und Exporte erfolgen und
- in welcher Form Daten und Informationen im Frontend dargestellt werden (Design).

Das KoGIs prüft dabei die Notwendigkeit der Installation einer gesonderten Entwicklungsumgebung (auf dem Entwicklungsserver FLORA) sowie die Abnahme der Entwicklung **vor Einbindung in die Produktivumgebung** und Onlinestellung.

Sollte die Entwicklung innerhalb einer gesonderten Entwicklungsumgebung notwendig sein, ist dem KoGIs ein grober Zeitplan vorzulegen, aus dem hervorgeht, wann die Umsetzung planmäßig beendet wird, so dass zu diesem Zeitpunkt ggf. eine Überprüfung der Performance und weiterer Kriterien erfolgen kann, bevor die Eigenentwicklung vom KoGIs freigegeben wird und damit auf das Produktivsystem übertragen werden kann.

Der benannte Zeitplan liefert auch einen Fertigstellungstermin, zu dem die Entwicklungsumgebung wieder gelöscht werden kann. Abweichungen von dem Zeitplan müssen eigenständig von den Dienststellen an das KoGIs-Team gemeldet werden.

Die durch die Qualitätsprüfung entstehenden Kosten - durch Beauftragung Dritter - werden in den Fällen, in denen Module allen anderen Anwendern zugutekommen, von dem Senator für Finanzen (KoGIs) getragen.

In den anderen Fällen sind die Kosten von den Dienststellen, Eigenbetrieben, Gesellschaften, Stiftungen und Einrichtungen zu übernehmen.

3. Integration von Eigenentwicklungen innerhalb der bereit gestellten KoGIs-Entwicklungsinstanz

Die Integration von Eigenentwicklungen erfolgt ausschließlich in den dafür vorgesehenen Ordnern (siehe Verortung der Eigenentwicklungen innerhalb der KoGIs-Instanz) und der dafür vorgesehenen Form.

Alle Anpassungen sind ausschließlich über das Backend des Six CMS vorzunehmen. Ein direkter Datenbank- oder Filezugriff auf den Servern wird in keinem Fall gewährleistet.

Die Dienststellen müssen diese Maßgabe bei allen Entwicklungen berücksichtigen.

4. Verortung der Eigenentwicklungen innerhalb der KoGIs-Instanz

Eigenentwicklungen dürfen ausschließlich unterhalb des Ordners **Eigene Erweiterungen** abgelegt werden.

Bei Templates, die unterhalb des Ordners **Site** von KoGIs eingebunden wurden, dürfen inhaltlich keine Änderungen vorgenommen werden.

Ausnahme 1: Bei der Integration von eigenen Inhalten in die lokale Suche kann es notwendig sein, bestimmte Abfragetemplates anzupassen. Diese Anpassung darf nur in Abstimmung mit dem KoGIs erfolgen und muss dem KoGIs gemeldet werden (siehe Meldeprozess). Damit wird ebenfalls sichergestellt, dass die abgestimmten Änderungen bei künftigen automatischen Aktualisierungen nicht überschrieben werden.

Ausnahme 2: Bei Verwendung von eigenen Dict-Einträgen sind diese in dem allgemeinen Dict-Ordner bzw. Dict-Container abzulegen. In allen Fällen sind eigene Dict-Einträge mit einem eigenen Dict-Pool zu versehen. So wird vermieden, dass bei Löschungen und Neuausspielen der Standard-Dict-Einträge die iegenen nicht ungewünscht gelöscht werden.

Diese Anpassung darf nur in Abstimmung mit dem KoGIs erfolgen und muss dem KoGIs gemeldet werden (siehe Meldeprozess).

Es sind keine weiteren Ausnahmen erlaubt.

5. Gestaltungsmöglichkeiten bei Verwendung des Projektmasters

Die Gestaltungsmöglichkeiten bei den Erweiterungen des Projektmasters sind genau dokumentiert (siehe Dokumentation Gestaltungsmöglichkeiten mit Hilfe der KoGIs-Basismodule sowie der erweiterten Projektmodule („Projektmaster“)). Abweichungen von diesen Vorgaben sind nicht erlaubt.

6. Sicherstellung der Übertragbarkeit

Da alle Webauftritte eigenen Instanzen (incl. eigener Datenbank) beinhalten, ist besonders die Übertragbarkeit der Module einzuhalten. Dazu zählen folgende Vorgaben:

- Modularer Aufbau der Module
- Festgelegte Nummernräume und Bezeichnungen bei der Namensvergabe der Templates
- Vermeidung sämtlicher statischer Adressierung
- Verwendung von GSIDs

6.1 Einhaltung des Nummernraums für Templates

Bei allen Eigenentwicklungen sind bestimmte Nummernräume vorgegeben und vollständig zu übernehmen:

- 00: Basiseinstellungen
- 01: HTML-Metainformationen und CSS
- 02: Header
- 03: Seitenaufbautemplates (ohne eigene Anzeige)
- 04: Menü
- 05: Content
- 06: Suchergebnis
- 07: Rechte Spalte
- 10: Zusammenfassende Templates
- 20: Seitentemplates (verfügbar in dem Seitencontainer)

6.2 Einhaltung der Benamung für Templates

Bei allen Eigenentwicklungen sind bestimmte Benamungen vorgegeben und vollständig zu übernehmen:

Die Endungen für Templates:

- _d für Detailtemplates
- _p für Seitentemplates
- _l für Listentemplates
- _q für Abfragetemplates

Notation: z.B. xx_namenskürzel_Funktion_der_Seite_d

Achtung: Templates, die nicht diesen Vorgaben entsprechen, werden zentral gelöscht.

***Ausnahme**: Seit der Six10-Version werden in KoGIs erstmals Trigger aus einem Template eingebunden. Diese Templates müssen genauso heißen wie die enthaltenen Funktionen oder Klassen. Da hierbei Six-seitig keine führenden Zahlen erlaubt sind, dürfen diese (und nur diese!) Templates von der Namenskonvention abweichen. Beispiel: trigger_pre_alias_d*

6.3 Einhaltung der Namenskonventionen für Workspaces, Rollen und Rechten

Die vorhandenen Workspaces, Rollen und Rechte dürfen ohne Absprache mit KoGIs nicht verändert werden.

Neu angelegte Workspaces, Rollen und Rechte dürfen eingerichtet werden, müssen allerdings namentlich herausstellen, dass es sich um eine Erweiterung handelt, Beispiel: Dienststelle X: Administrator.

Achtung: Filter und Workspaces, die nicht diesen Vorgaben entsprechen, werden zentral gelöscht.

6.4 Vermeidung statischer Adressierung und Verwendung der GSId

Um die 100%ige Übertragbarkeit der Entwicklungen zu gewährleisten, ist darauf zu achten, dass statische Adressierungen vollständig vermieden werden.

Bei den Entwicklungen muss darauf geachtet werden, dass die Container nicht mit den IDs sondern ausschließlich über Pfade angesprochen werden. Beispiele:

```
$out = Array('container' => 'Site: News');
$counter = new Container("Site: Weitere Container: Counter");
$news_area_id = find_containerID("Site: News");
```

Achtung: Templates, die nicht diesen Vorgaben entsprechen, werden zentral gelöscht.

Zusätzlich wird bei allen Artikeln, Containern und Templates die globale System ID (GSId) verwendet.

7. Sicherstellung der Qualität und Performance

Um die Qualität und die Performance sicher zu stellen, **werden vor Onlinestellung** analog zur Nutzung der regulären KoGIs-Basismodule Reviews erstellt und kostenlos Beratungsleistungen zu allen Querschnittsaufgaben bereitgestellt.

Für das Review werden folgende Informationen benötigt[1]:

1. eine Übersicht der neuen und angepassten Templates
2. eine Übersicht der neuen und angepassten Containerstrukturen
3. eine Übersicht der neuen Scheduler (Anpassungen bestehender Scheduler sind nicht erlaubt)
4. eine Übersicht der neuen und angepassten Filter und Workspaces

Die notwendigen Daten sind in der beigefügten Beispieldokumentation vollständig zu liefern.

Die Überprüfung der Entwicklung (bei der initialen Überprüfung und jeder Folgeprüfung) erfolgt in der Regel innerhalb von zwei Kalenderwochen (Sie sollten also immer genügend Zeit einplanen, damit diese Schritte erfolgen können).

Die durch die Qualitätsprüfung entstehenden Kosten - durch Beauftragung Dritter - werden in den Fällen, in denen Module allen anderen Anwendern zugutekommen, von dem Senator für Finanzen (KoGIs) getragen. **In den anderen Fällen sind die Kosten von den Dienststellen, Eigenbetrieben, Gesellschaften, Stiftungen und Einrichtungen zu übernehmen.**

Werden die Kriterien für Performance, Last, Design usw. nicht eingehalten, hat KoGIs das Recht, die Eigenentwicklung zur Überarbeitung zurückzuweisen – dies kann auch mehrfach erfolgen (Sie sollten also immer genügend Zeit einplanen, damit diese Schritte erfolgen können). Die Folgekosten durch den Mehraufwand sind von den Dienststellen, Eigenbetrieben, Gesellschaften, Stiftungen und Einrichtungen zu übernehmen.

[1] Sollten keine Anpassungen in diesen Bereichen vorgenommen worden sein, ist dies ebenfalls anzugeben.

8. Sicherstellung der Barrierefreiheit

Die KoGIs-Basismodule werden kontinuierlich durch das Kompetenzzentrum Barrierefreiheit beim Institut für Informationsmanagement GmbH in Hinblick auf die „Bremer Barrierefreie Informationstechnik-Verordnung“ (BremBITV) überprüft und die Ergebnisse in Softwareevaluationen zusammengefasst.

Die Einhaltung der BremBITV stellt auch bei Eigenentwicklungen eine zwingende Voraussetzung dar, so dass alle Eigenentwicklungen, die im Frontend einsehbar sind, eine Kurzreviewerstellung in Hinblick auf die BremBITV bzw. den BITV-Test erforderlich machen. Die Ergebnisse sind dem KoGIs zeitnah vor Onlinestellung vorzulegen.

Werden die Kriterien für Barrierefreiheit nicht eingehalten, hat KoGIs das Recht, die Eigenentwicklung zur Überarbeitung zurückzuweisen.

9. Weitere Empfehlungen für die Entwicklung

9.1 Namensräume

Innerhalb von SixCMS sollen Namensräume an verschiedenen Stellen verwendet werden:

9.1.1 Feldnamen

Die Kombination aus Feldname und Feldtyp muss innerhalb einer Six-CMS Instanz eindeutig sein. Es ist nicht möglich, etwa ein Sprachfeld, dass in einer Entwicklung als *language* bezeichnet wird und den ISO-Code der Sprache enthält, in einer anderen Entwicklung als Relationsfeld mit gleichem Namen zu verwenden. Um dieses Problem zu entschärfen, wird empfohlen, zumindest für Linkfelder und Relationsfelder eine Präfixnotation zu verwenden, die den Datentyp des jeweiligen Feldes anzeigt. Linkfelder enthalten damit den Präfix *lnk* und Relationsfelder des Präfixes *rel*. Es ist unbedingt darauf zu achten, dass bei Relationsfeldern auch das jeweilige Feld im korrespondieren Container dieses Präfix erhält.

Neben dieser eher technischen Separierung in Namensräume sollte zusätzlich eine logische Separierung verwendet werden. Dies bedeutet, dass alle Entwicklungen einen für diese Entwicklung eindeutigen Namensraum für die Feldbezeichnungen zugewiesen bekommen. Dieser Namensraum ist dann als Präfix oder Postfixnotation zusätzlich zum gewünschten Feldnamen zu modellieren. Es hat sich bewährt, einen 1-3-stelligen Code zu verwenden. So würde das schon oben genannte Textfeld *language* als Textfeld im Namensraum *afz* mit *afz language* und als Relationsfeld im gleichen Namensraum mit *afz rel language* bezeichnet werden.

9.1.2 Templatenamen

Die Bezeichnung von Templates (label) muss innerhalb einer SixCMS Instanz ebenfalls eindeutig sein. Auch hier empfiehlt sich also der Einsatz eines Präfixes oder Postfixnotation analog der Bezeichnung der Felder.

9.1.3 Label System Identifier (LSID)

In SixCMS 7.1 wurde als weiteres Identifikationsmerkmal für Content (neben der numerischen ID und der alphanumerischen GSID) eine alphanumerische LSID eingeführt. Ähnlich wie bei Templates unterliegen auch LSID dem globalen Zwang zur Eindeutigkeit. Es kann also innerhalb einer SixCMS Instanz keine zwei Datensätze mit gleicher LSID geben. Insbesondere bei Import und Replikation kann das zu Problemen führen, da ja neben der Eindeutigkeit der GSID auch eine Eindeutigkeit der LSID gefordert wird. Im Gegensatz zur GSID wird aber die LSID manuell vergeben, was die Fehleranfälligkeit erhöht. Es wird daher empfohlen, die LSID eher sparsam und nicht abweichend von ihrem vorgesehenen Verwendungszweck einzusetzen. Auch die Verwendung von Namensräumen in einer Präfix oder Postfixnotation erscheint angeraten.

Da es verschiedene Einsatzbereiche für die LSID gibt, sollten diese Einsatzbereiche ebenfalls durch Namensräume abgegrenzt werden. Zu diesen Einsatzbereichen zählen:

- Referenzieren von Schmuckbildern in Templates – Hier könnte ein Namensraum pic benutzt werden
- Referenzieren von CSS Dateien – Hier könnte ein Namensraum css benutzt werden
- Verwendung der LSID als Key für einen Mehrsprachigkeitsansatz – Hier könnte ein Namensraum dct benutzt werden, dennoch sollte diese Technik nicht überstrapaziert werden, da immer auch auf die Performance zu achten ist. Eine Alternative bietet sich durch die Verwendung von gettext an.
- Verwendung der LSID als Ersatz für die ID oder GSID in URL's, um besser lesbare URL's zu erreichen. Generell ist diese Verwendung in mehrsprachigen Umgebungen als eher problematisch anzusehen, da dann auch die LSID der verwandten Sprachdatensätze übersetzt werden müssen und mit Konflikten zu rechnen ist, wenn Systeme zusammengeführt werden sollen.

9.1.4 Variablen

PHP-Variablen werden in Templates an verschiedenen Stellen verwendet, wenn zusätzliche Logik erforderlich ist, die mit der SixCMS Templatesprache nicht abgebildet werden kann. Alle Variablen haben einen Gültigkeitsbereich (Scope). Dieser ist innerhalb eines eingebetteten Templates immer lokal, da aus technischer Sicht ein eingebettetes Template eine PHP Funktion darstellt. Es wird empfohlen, eigene Variablen mit einem Unterstrich beginnen zu lassen, um potentielle Konflikte mit evt. verwendeten gleichnamigen globalen Variablen zu vermeiden. Generell sollte auf globale Variablen verzichtet werden. Insbesondere zur Kommunikation von Templates untereinander sind sie wegen der potentiellen Seiteneffekte nicht sinnvoll einzusetzen. Eine Kommunikation von Templates untereinander sollte immer über definierte Zugriffspunkte einer selbst definierten Klasse erfolgen. Im folgendes ist ein Beispiel für eine solche Klasse dargestellt:

```
class my_cms{
/**
* statische variable zum halten eines lokalen
* namensraumes
*/
private static $merke_ary;
/**
* setter methode zum merken eines wertes
*/
      public static function set($label, $value){
```

```
        self::$merke_ary[$label] = $value;
    }
    /**
    * getter methode zum abrufen eines wertes
    */
        public static function get($label){
        return isset(self::$merke_ary[$label]) ?
        self::$merke_ary[$label] :
        null;
       }
    }
```

Diese Klasse hält in einer statischen Variable, die als assoziatives Array ausgelegt ist, beliebige Werte, die auf einen Request bezogen sind. Werte können von beliebiger Stelle im Code gesetzt und wieder abgerufen werden. Die Variablen sind nicht global und verschmutzen so nicht den globalen Namensraum und sie sind über einen definierten Zugriffspunkt setzbar und wieder abrufbar. Global ist in dieser Implementierung lediglich der Name der Klasse, hier sollte also eine Namensraumkonvention verwendet werden. Für die Variablen selbst ist dann keine Namenskonvention mehr notwendig. Es empfiehlt sich jedoch zu Dokumentationszwecken, die verwendeten Label innerhalb der Klasse zu hinterlegen. Auch das Setzen von Werten aus den GET und POST Variablen kann auf diese Weise über einen einheitlichen Zugriffsmechanismus geregelt werden. So kann in Templates ein Zugriff auf die superglobalen Werte vermieden werden, wodurch die Portabilität von Templates verbessert wird.

9.1.5 Sessions

Die Verwendung von serverseitigen Sessions ist ein beliebtes Mittel, um Daten requestübergreifend in Anwendungen zur Verfügung zu haben. Aus technischen Gründen ist es in einer Anwendung jedoch sehr schwierig, mehrere voneinander unabhängige Session parallel zu verwalten. Daher werden i.d.R. gemeinsam verwendete Sessions benutzt.

Dies bedeutet, dass mehrere Anwendungen die gleiche physikalische Session benutzen müssen. Daher gelten hier sinngemäß die gleichen Empfehlungen, wie sie schon für Variablen ausgesprochen wurden. Speziell auf Sessions bezogen können folgende Empfehlungen hilfreich sein:

- Sessions sollten sparsam verwendet werden, falls möglich sollte auf sie verzichtet werden
- werden Sessions benutzt, sollten diese logisch in Namensräume separiert werden. SixCMS verwendet als Namensraum cms sess mit einer Prüfsumme, dieser Namensraum darf also keinesfalls verwendet werden.
- Die Menge der in einer Session gespeicherten Daten sollte so klein wie möglich gehalten werden. Sessiondaten werden bei jedem Request geparst und verarbeitet, was die Performance negativ beeinflusst.
- Sessions sollten so schnell wie möglich wieder geschlossen werden, da aus technischen Gründen immer nur genau ein Webserverprozess auf die gleiche Session zugreifen kann. Insbesondere bei der Verwendung von Frames, Ajax und ähnlichen Techniken kann dies zu einem Problem werden (Frames sollten ja eh´ nicht verwendet werden ☺).
- Soll die SixCMS interne Session mit verwendet werden, sollte die Funktion cms session start() statt der PHP internen Funktion session start benutzt werden. Hierdurch werden einheitliche Systemeinstellungen für die Session erreicht.

- Auf Sessionvariablen sollte innerhalb von Templates nie direkt sondern immer vermittels einer kleinen Wrapperfunktion zugegriffen werden. So reduzieren sich die Stellen bei einer potentiellen Änderung von Namensräumen usw.

10. Konsequenzen und Ausblick auf zukünftige Weiterentwicklungen (auch der KoGIs-Basismodule)

Für die Entwicklung innerhalb von SixCMS und neben den KoGIs-Basismodulen ist neben technischen Randbedingungen auch eine Reihe von organisatorischen Voraussetzungen zu erfüllen, damit eine verteilte Arbeit an einem Gesamtsystem erst möglich wird. Hierzu zählen die besprochenen Namensräume, die auf möglichst allen Gebieten einzuhalten sind.

Daneben sollten einige Designprinzipien verwendet werden, die maßgeblich auf die Modularisierung der einzelnen Komponenten untereinander abzielen. Hierzu zählen:

- Verwendung von Linkfeldern der Verwendung von Relationsfeldern vorziehen. Linkfelder sind im SixCMS unidirektionale Links von einem Datensatz auf einen anderen Datensatz. Der Zieldatensatz wird dabei nicht verändert. Durch die automatische Linkverwaltung innerhalb von Six-CMS werden beim Löschen oder Offline Stellen des Zieldatsatzes auch die Links im Quelldatensatz im Frontend nicht mehr verwendet. Linkfelder können zudem mit Artikeln aus beliebigen Containern verwendet werden. Bei der Verwendung von Relationsfeldern wird auch das Datenmodell des Zieldatensatzes mit verändert. Dadurch kann etwa die Performance bei Anwendungen, die alle Relationen des Zieldatensatzes auslesen, negativ beeinflusst werden.
- Klassenorientiert statt prozedural arbeiten. Die Überlegenheit einer objektorientierten Programmierung ist heute gegenüber einer prozeduralen Programmierung unbestritten. Durch die Möglichkeit einzelne Methoden zu überladen bestehen viel bessere Möglichkeiten, vorhandenen Code mehrfach zu verwenden. Zudem wird das Namensraumprinzip erleichtert. Statt also z.B. einen eigenen SixCMS Platzhalter als Funktion zu implementieren kann dieser besser als statische Methode einer selbst geschriebenen Klasse implementiert werden.
- Schnittstellen für Daten und Funktionen vorsehen. Bei der Entwicklung von SixCMS Anwendungen sind in einem frühen Designstadium Schnittstellen für andere Module innerhalb der gleichen oder einer anderen SixCMS Instanz vorzusehen. Dabei können Schnittstellen sowohl in Form von Funktionen die Daten liefern implementiert werden, als auch komplette HTML Schnipsel bereitgestellt werden. Auch die Bereitstellung von Ajax-Elementen ist so möglich. Es sollte darauf geachtet werden, dass eine sinnvolle und konsistente Schnittstellendokumentation erstellt wird, die die Implementierungsdetails vor dem Benutzer der Schnittstelle versteckt. So können später Änderungen innerhalb der Anwendungen erreicht werden, ohne Drittanwendungen, die diese Schnittstelle verwenden, modifizieren zu müssen.
- Konvention geht vor Konfiguration. Dieses anerkannte Designprinzip aus der ruby on rails Welt kann auch für SixCMS angewendet werden. So kann etwa per Konvention geregelt werden, ob ein Template in einer konkreten Instanz durch ein anderes Template ersetzt werden kann. Wird als Basistemplate z.B. afz_teaser_d verwendet, kann dieses per Konvention durch ein Template mit dem Label afz_teaser_b40_d ersetzt werden. b40 ist hierbei etwa ein in eindeutiger Weise (z.B. aus dem Lizenzidentifier) ableitbarer Suffix.
- Topdown statt seriell arbeiten. Oft sind in der Entwicklung von SixCMS Templates Konstruktionen zu finden, die die zwingende Verwendung zueinander passender Templates voraussetzen. Ein typisches Beispiel sind Header und Footer Templates einer Seite. Hier werden in einem Header-Template HTML Tags geöffnet, die in einem anderen Template (Footer) erst wieder geschlossen werden. Dieser Ansatz führt zu einer hohen Komplexität

des Templatesystems, da die beiden Templates voneinander abhängig sind (was nicht sinnvoll im SixCMS modelliert werden kann) und ihre separate Wiederverwendung erschwert. Besser ist es, Templates zu entwerfen, bei denen jedes öffnende Tag im gleichen Template auch wieder geschlossen wird.

11. Aktualisierungen

Datum	Thema	Art	Seite
März 2020	Löschung	Kapitel 3: Entfernen eines Absatze zu dem Verbot der Verwendung von Triggern und Custom-Classes	6
	Löschung	Kapitel 10: Entfernen des Ausblicks der Integration von Triggern und Custom-Classes	18
November 2019	Aktualisierung	Kontaktdaten aktualisiert: E-Mail und Dienststelle	1, 2, 5, 11
Februar 2019	Ergänzung	Kapitel 6.2: Ausnahme der Namenskonvention für Trigger in Templates	9
	Aktualisierung	Kontaktdaten	2
	Löschung	Kapitel 2: Grafik mit Serverinfrastruktur entfernt	5
Juni 2015	Aktualisierung	Grundsätzliche Überarbeitung	alle
April 2011	Neu erstellt		alle