Projektdokumentation des Abschlussprojekts

mindworks Portal
mit relativen
Artikeln

Fachinformatiker Fachrichtung Anwendungsentwicklung
Abschlussprüfung 2014
Eine Arbeit von
Jens Beyer

Inhaltsverzeichnis

1. Ist-Analyse, Ziel des Auftrags

Das Unternehmen mindworks möchte seine Webseite erneuern. Die Einpflege von Artikeln bereits bestehender Projekte, soll mit einem Backend Bereich realisiert werden. Um eine möglichst zielgerichtete suche im mindworks Portal zu ermöglichen werden Tags genutzt.

1.1 Ist-Zustand

Die Projekte der Firma werden durch eine Homepage mit Slider repräsentiert.

Es herrscht eine einheitliche Navigation mit einem Slider auf der Homepage.

Beispiel auf der Seite www.mindworks.de. An der Seite wird nach besuchtem Artikel aus dem Slider die Homepage neu aufgebaut und man kann auf der linken Seite ein Baumartiges Menü sehen. Der folgende Screenshot verdeutlicht dies.

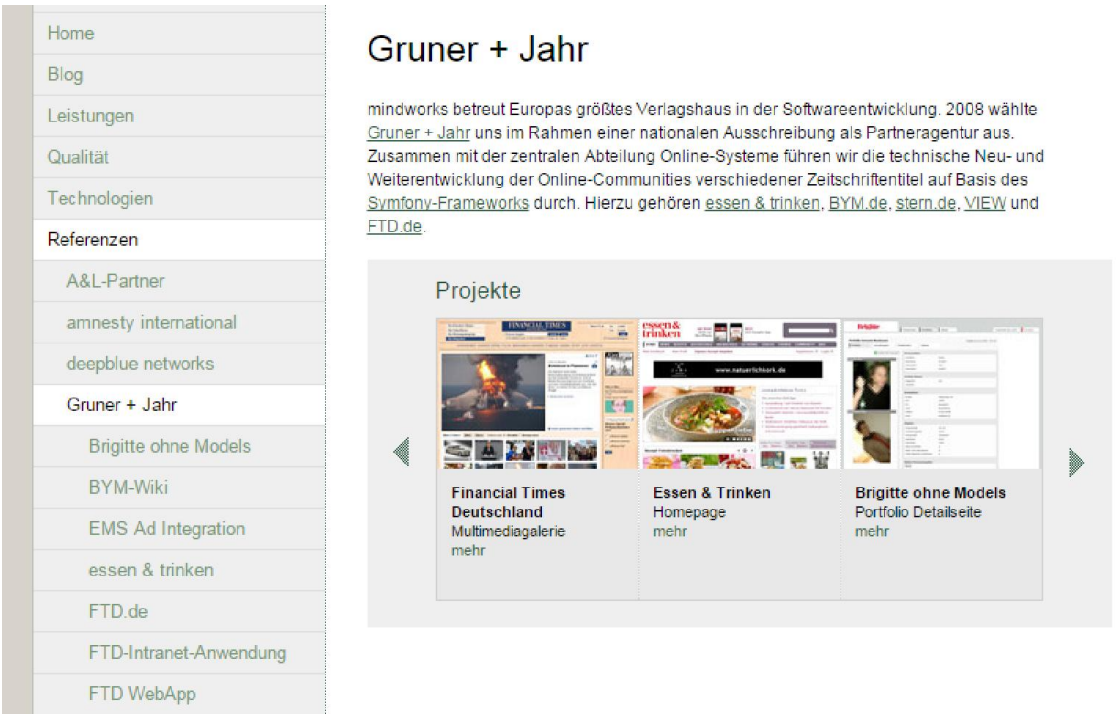


Abbildung 1: Foto : mindworksHomepage.png: [28.11.2014, Jens Beyer]

Die Seite ist mit einfachen PHP-Mitteln aufgebaut worden.

1.2 Soll-Zustand

Entwicklung einer Datenbank zur Verwaltung und Bereitstellung von Text- und Bildinformationen zu Kundenprojekten der mindworks GmbH sowie deren Beziehungen untereinander.

Im Gegensatz zu einer üblichen Baumstruktur, in der Referenzkunden, Kundenprojekte und eingesetzte Technologien relational verknüpft werden, wünscht sich die Geschäftsführung eine Verknüpfung der Inhalte über Schlagworte, sogenannte Tags.

Auf diese Weise sollen sich Beziehungen gewichten lassen und weitere Medien wie Logos, Screenshots und Fotos aus der Projektarbeit beliebig auf Anwenderebene hinzufügen lassen.

Das Frontend zur Anzeige soll im Rahmen dieses Projektes explizit nicht entwickelt werden. Benötigt wird jedoch eine einfache Ausgabe, um die grundsätzliche Funktionalität zur Bereitstellung der Inhalte zu demonstrieren.

Auf Wunsch der Geschäftsführung sollen für die Realisierung der Anwendung zwingend PHP und MySQL eingesetzt werden.

Das Framework der Wahl ist Symfony oder Phalcon.

2. Nutzen für den Kunden und das Unternehmen

Der Nutzen für das neue Portal mit relativen Artikeln wird hier für den Kunden, als auch dem Unternehmen, erläutert. Die einpflege der Artikel mit der neuen Struktur erweist sich als effizient.

2.1 Nutzen für den Kunden

Kunden bekommen einen besseren und zielgeführten Einblick in die Projekte und Arbeit von mindworks. Dies gibt mehr Vertrauen schon von Anfang an. Kunden bleiben länger auf der Webseite, empfehlen sie eventuell sogar weiter. Die richtigen Kunden fragen an, da eine Fülle von ähnlichen Kunden schon mit der Firma gute Erfahrung gemacht hat und vielleicht auch ein zugeschnittenes Projekt schnell vom Klienten ersichtlich ist.

2.2 Nutzen für das Unternehmen

Das Unternehmen präsentiert sich auf der Homepage durch Struktur und Prioritäten, die dem Kunden schon gleich bei der Suche ins Auge fallen. Dadurch wird ein bleibender Eindruck erzeugt und die Seite auch weiter empfohlen.

Das Unternehmen bekommt dadurch zielorientierte Angebote, auf die es eingehen kann. Somit wird eine Kundennähe gezeigt und die Stärken der Firma in den Vordergrund gestellt.

2.3 Wirtschaftlicher Nutzen

Der wirtschaftliche Nutzen ist nicht messbar. Nur über die Zeit wird sich einstellen ob die Suchorientierte Präsentation von zusammen hängenden abgeschlossenen Firmenprojekten die Erfolgsquote verbessert und dem Kunden auffällt.

Weitere Aspekte wären unnötige Zeitaufwendung für Kunden, die sich nicht im Interesse oder der Umsetzung auszahlen.

Die Projekte werden meist auf Messen und Business Portalen durch Kunden angefragt. Die Kosten für entfernte Messen beinhalten daher die Bahnfahrt und evtl. auch Aufenthalt vor Ort. Diese Kosten können verringert werden durch einen bleibenden Eindruck auf der Homepage von mindworks.

2.3.1 Alternatives System Kostenkalkulation

Die relativ zueinander stehenden Artikel könnten auch mit einem Content Management System realisiert werden. Dies ist erheblicher Aufwand zumal die Beziehungen komplexer werden.

Bei der Kalkulation mit dem CMS werden die Minuten für das erstellen von Seiten und deren Abhängigkeiten auf das Maximum berechnet.

Wurzel Seite 1,2,3

Klick 1 → Seite 1 mit 2,3 → Klick auf 2 → Seite 2 mit 3

Klick 2 → Seite 2 mit 1,3 → Klick auf 1 → Seite 1 mit 3

Wenn zum Beispiel die Wurzelseite angezeigt wird, wird Artikel 1 mit 2,3 als relativen Artikel angezeigt. Dann wird nach Klick auf 2 der neue Seitenaufruf, 2 mit relativen Artikel 3, da eins ja schon besucht worden ist, angezeigt. Wenn man wieder auf der Wurzel ist und wiederum den

Artikel 2 mit 1,3 als relativen Artikel anklickt, würde nach Klick auf Artikel 1 der neue Seitenaufruf, Artikel 1 mit relativen Artikel 3, folgen.

Die Abhängigkeiten resultieren aus den Seitenerstellungen der oben genannten Möglichkeiten. Hierzu wurde ein rekursiver Algorithmus benutzt, um die Abhängigkeiten zu berechnen.

Eine Kalkulation der wachsenden Anforderungen mit einem CMS wird hier aufgezeigt.

seite einrichten	kleinste Abhängigkeit	größte Abhängigkeit	Abhängigkeiten einrichten einer Seite in m	gesamt größte Abhängigkeit m
1	0	0	4	4
2	1	2	4	8
3	1	7	4	28
4	1	16	4	64
5	1	30	4	120

Abbildung 2: Tabelle : CMS Kalkulation in Minuten [03.12.2014, Jens Beyer]

Die Umsetzung im Projekt hingegen basiert auf Editieren von Tags, Artikel und Kunden mit deren Relationen. Für einen neuen Artikel wurde die gleiche Zeit genommen wie beim CMS. Die Abhängigkeiten sind geringer und wachsen nicht exponentiell bei hinzufügen eines neuen Artikels.

Eine Kalkulation der wachsenden Anforderungen mit dem neuen Portal wird hier aufgezeigt.

seite einrichten	kleinste Abhängigkeit	größte Abhängigkeit	Abhängigkeiten einrichten einer Seite in m	gesamt größte Abhängigkeit m
1	0	0	4	4
2	1	2	4	8
3	1	3	4	12
4	1	4	4	16
5	1	5	4	20

Abbildung 3: Tabelle : neues Portal Kalkulation in Minuten [03.12.2014, Jens Beyer]

Wenn man dies auf die Zeit eines Mitarbeiters rechnet, würden folgende Kosten entstehen.

seite einrichten	Differenz in Minuten	Kosten pro Stunde	Differenz Preis
1	0	35	0
2	0	35	0
3	16	35	9,3333333333
4	48	35	28
5	100	35	58,3333333333

Abbildung 4: Tabelle : Projekt Kalkulation pro Mitarbeiter Gehalt [03.12.2014, Jens Beyer]

Fazit: Bei 5 Seiten und größtmöglicher Abhängigkeit ist die Zeit der Umsetzung der neuen Portal Struktur schon wieder eingespart.

3. Projektabgrenzung

3.1 Zeiterfassung

Bei der Zeiterfassung werden die Analyse und Mehrwege nicht betrachtet. Es wurde weitaus mehr Aufwand betrieben um die Optionen zu betrachten. Diese Optionen werden klar abgegrenzt und sind nicht in der Zeiterfassung enthalten.

3.2 Anforderung und Ziele des Projekts

3.2.1 Projektübersicht

3.2.1.1 Systemvoraussetzungen

Betriebssystem: Linux Debian v7.4

Framework: Symfony v2.5.7

Entwicklungssprache: PHP v5.4

Webserver: Apache/2.4.9 (Debian)

Doctrine: v1.3.0

MySQL: v5.5.35-2

Test Tool: PHPUnit v.4.0

3.2.1.2 Abgrenzung

Bei der folgenden Übersicht wird die Betrachtung der Optionen aufgezeigt.

Das Projekt wird mit dem Framework Symfony 2 umgesetzt.

In diesem Framework gibt es viele Erweiterungen, die für dieses Projekt in Betracht gezogen werden können. In Firmenprojekten wird als Backendlösung für das Einfügen von Inhalten ein bekanntes Bundle, eine Erweiterung benutzt. Es ist das Sonata Bundle. Welches eine Adminumgebung bereitstellt und Inhalte löschen, editieren wie auch erstellen kann. Die Dokumentation von diesem Bundle ist sehr knapp gehalten. Die Einsetzung wurde in Betracht gezogen und stellte sich im Nachhinein als optional heraus, Da die Inhalte langsamer im Backend geladen werden.

Aus diesem Grund wird eine eigens verwendete Struktur als Lösung verwendet, die auch verständlich und leicht erweiterbar ist.

In Symfony gibt es mehrere Möglichkeiten um Ziele umzusetzen. Unter den sogenannten Form Klassen, die auch in HTML unter dem <form> bekannt sind, gibt es mehrere Verwendungen um die Inhalte zu laden und zu speichern. Hierbei wird nicht auf die Javascript seitige Variante eingegangen, die zwar dynamisch ist, dennoch mit mehr Komplexität einhergeht. Die Trennung von Zuständigkeiten „seperation of concerns“ wurde in der Projektbeschneidung beachtet und somit keine dynamische Manipulation (insert, update, delete) von verschachtelten Abhängigkeiten mit Javascript in Erwägung gezogen.

3.2.2 Definitionen von Inhalten

Bei den definierten Inhalten steht die Beschneidung auf die Kerninhalte des Projekts im Vordergrund.

Die erwähnten Inhalte in der Projektübersicht haben durchaus mehr Zeit in Anspruch genommen um sie abzuwägen und zu testen. Auch die langwierige Evaluation der Doctrine Komponente, dem Objekt orientierten Relations-Modell, auf Datenbank Ebene wird nicht mit in die Zeitaufwendung hinein genommen.

Das bereitstellen von Text Inhalten und Bildern wurde durch die begrenzte Zeit weggelassen. Dazu hätte man die Einbindung von dem CKE Editor in Betracht ziehen und die Datenstruktur dementsprechend anpassen müssen.

Die dokumentierten Inhalte beziehen sich daher nur noch auf die Kerninhalte des Projekts.

4. Projektarbeit-Kerninhalte

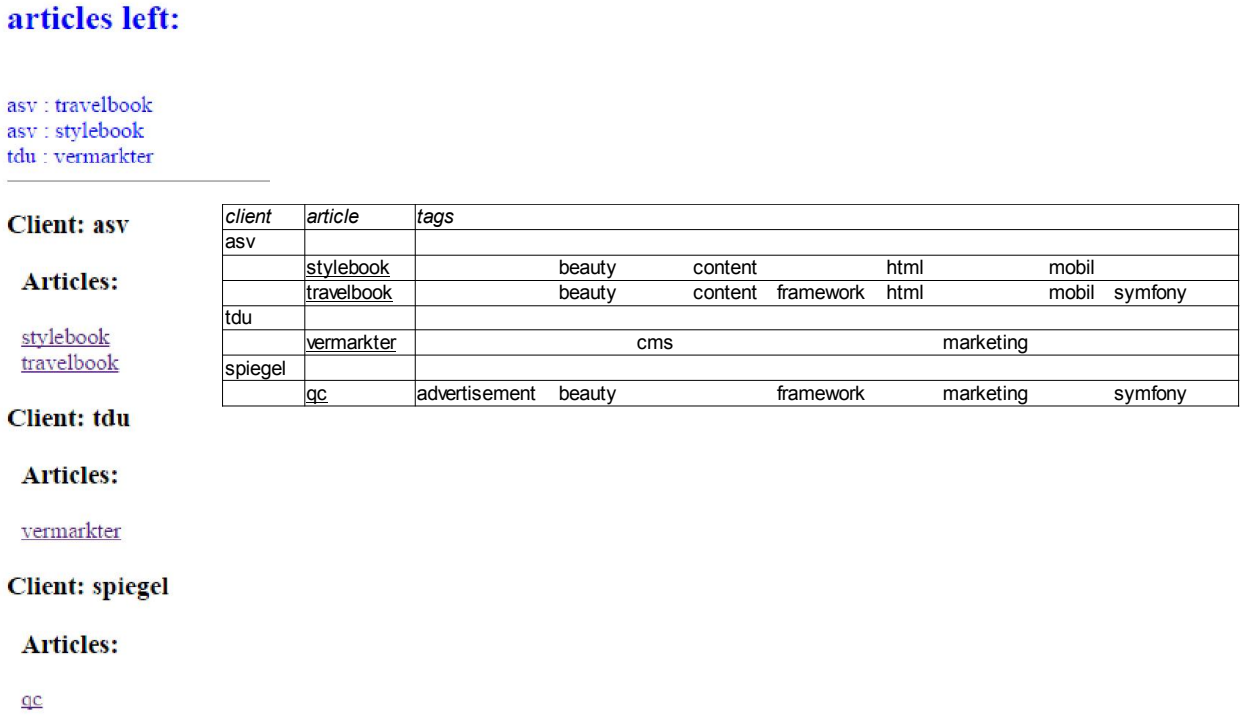
4.1 Kerninhalte

Die Übersicht der Elemente und Funktionen des Projekts wird aufgezeigt.

Hierbei wird vom Frontend auf das Backend hingeführt.

4.1.1 Frontend

Die Ansicht des „Portals“ wird mit allen vorhandenen Links zu Artikeln mit folgender Struktur dargestellt. Die rechte Tabelle ist eine Hilfstabelle um die Relationen zu veranschaulichen.



client	article	tags						
asv								
	stylebook		beauty	content		html	mobil	
	travelbook		beauty	content	framework	html	mobil	symfony
tdu								
	vermarkter			cms			marketing	
spiegel								
	qc	advertisement	beauty		framework		marketing	symfony

Abbildung 5: Foto : PortalShow.PNG [03.12.2014, Jens Beyer]

Hierbei ist eine baumartige Struktur zu erkennen. Damit die Funktion des Projekts präsentiert werden kann, werden alle Artikel im unteren Bereich gezeigt und oben die sichtbaren Artikel für den Besucher der Webseite (hier wurde auf qc geklickt). Also hat der Artikel qc relevante Tags zu den anderen oben gezeigten Artikeln.

Hier kann man auch die Relationen mit den Tags in der Tabelle erkennen.

Im Anhang wird auf weiteres Verhalten eingegangen und die Führung des Webseiten Besuchers durch die Artikel auf dem mindworks Portal.

Um Tags, Artikel und Kunden einzupflegen wird folgende Umgebung benutzt.

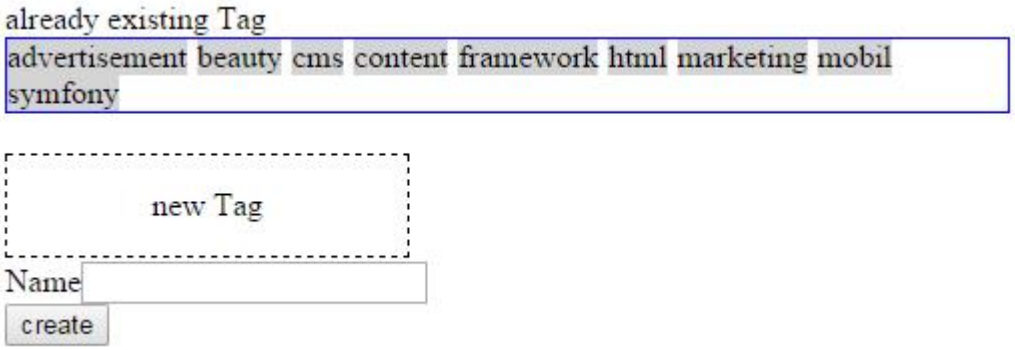


Abbildung 6: Foto : tagNew.PNG [03.12.2014, Jens Beyer]

Abbildung 7: Foto : tagShow.PNG [03.12.2014, Jens Beyer]

Abbildung 8: Foto : articleNew.PNG [03.12.2014, Jens Beyer]

Abbildung 9: Foto : articleShow.PNG [03.12.2014, Jens Beyer]

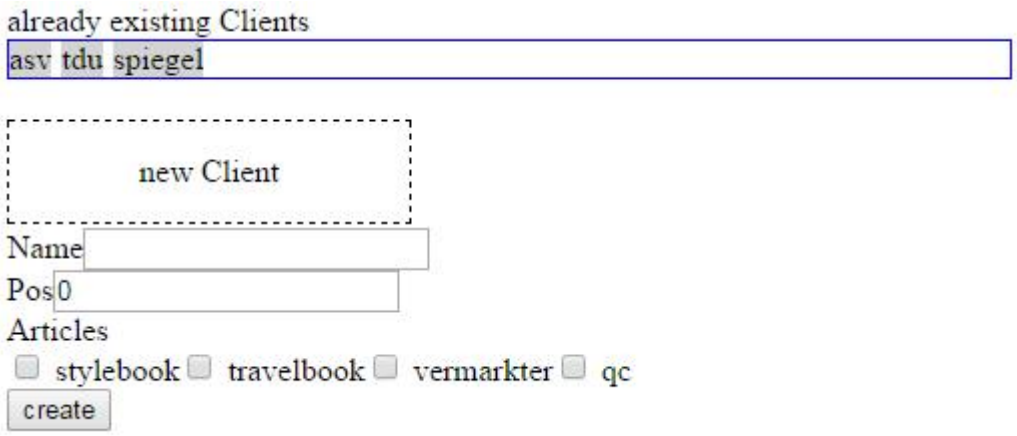


Abbildung 10: Foto : clientNew.PNG [03.12.2014, Jens Beyer]

Abbildung 11: Foto : clientShow.PNG [03.12.2014, Jens Beyer]

4.1.2 Backend

Das Backend wird aus folgenden Bestandteilen realisiert.

Controller, PortalData Klasse, Datenbankzugriffe (Entitäten, Repositories)

4.1.2.1 Controller

Der Controller stellt die Ausführung einer Funktion von einer besuchten URL bereit. Um die Daten Kunde, Artikel und die dazugehörigen Tags pflegen zu können, existieren Kontextabhängige URLs. Die Controller Funktionen unterscheiden sich in „show“, „delete“, „create“, und „edit“ Funktionalität im Backend Pflege Bereich. Für die Ausgabe der besuchten Artikel gibt es auch noch eine „show“ und „visit“ Funkton, die im PortalController lagern.

4.1.2.2 Facade

Das „FacadeControllerInterface“ entkoppelt das Framework und beinhaltet Konventionen für die Benutzung der Struktur. Es wird auf die Controller angewendet, die bei erstellen eines Controller Objekts die Funktion setFacade(....) aufrufen. Im Aufruf setFacade wird das objektorientierte Doctrine gekapselt. Dies ist sehr nützlich für Testzwecke. Somit werden Aufrufe zur Datenbank in einer Klasse namens „RepositoryFacade“ beschränkt.

4.1.2.3 PortalData Klasse

Die PortalData Klasse beinhaltet auch ein facade Attribut.

Wenn Die PortalData Klasse im PortalController erstellt wird muss ein Facade Objekt von außen (hier vom Controller) an die Klasse übergeben und gesetzt werden. Dadurch ist der Aufruf der FacadeRepository Klasse wieder gekapselt.

Die PortalData Klasse enthält eine eigene Logik, die mit der Session schon besuchte Artikel rausfiltert und die Daten bei jedem Aufruf aktualisiert und vom Controller abgefragt und angezeigt werden kann. So ist gewährleistet, dass der Benutzer oder Besucher der Webseite durch nicht besuchte Links bzw. Inhalte durch navigiert werden kann.

4.1.2.4 Datenbankzugriffe

Die Datenbankzugriffe werden durch Entitäten gesteuert.

Hierbei handelt es sich um 2 Komponenten. Die erste ist die Notation mit Doctrine um die Datenbank Struktur zu erstellen. Die 2. ist die Verwendung von PHP Code um die Objekte der Datenbank abzubilden. Ein ERM-Diagramm ist hier dargestellt.

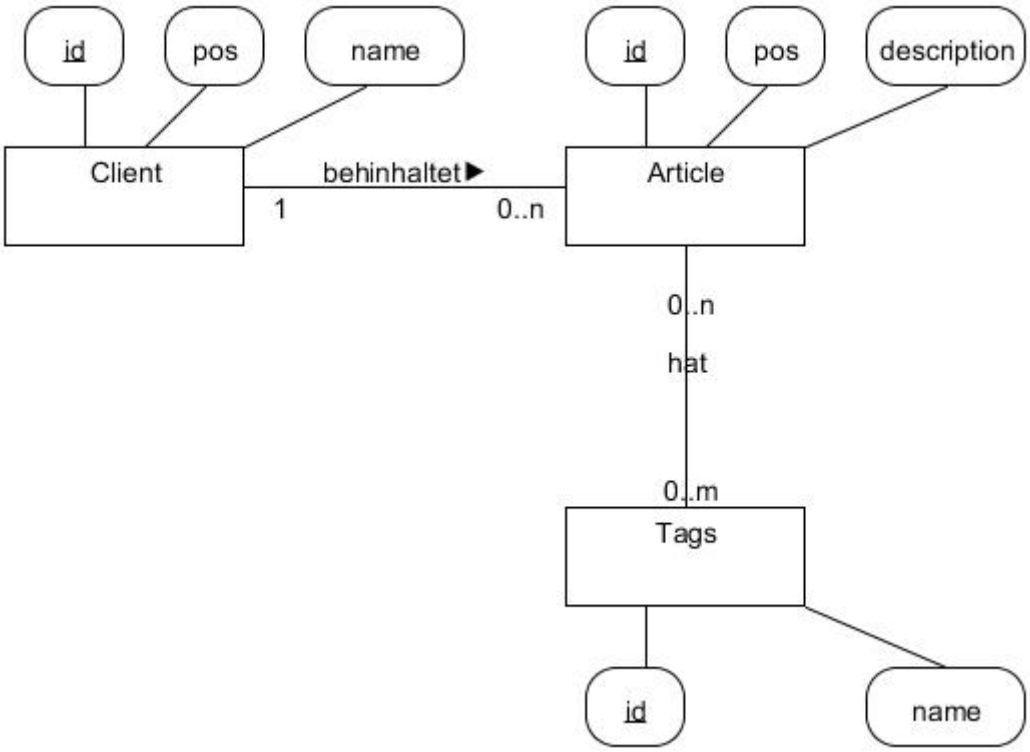


Abbildung 12: Foto : Entitys.png

Ein Ausschnitt vom PHP Code zeigt die Abhängigkeiten der Objekte zueinander.

Client.php

```
public function __construct()
{
  $this->articles = new ArrayCollection();
}
public function addArticle(Article $article)
{
  $article->setClient($this);
  $this->articles->add($article);
}
public function removeArticle(Article $article)
{
  $this->articles->removeElement($article);
}
```

Article.php

```
public function __construct()

{
  $this->tags = new ArrayCollection();
}
public function setClient(Client $client)
{
    $this->client = $client;
}
public function removeClient()
{
  $this->client = null;
}
public function addTag(Tag $tag)
{
  $tag->addArticle($this);
  $this->tags->add($tag);
}
public function removeTag(Tag $tag)
{
  $this->tags->removeElement($tag);
}
```

Tag.php

```
public function addArticle(Article $article)
{
  if (!$this->articles->contains($article)) {
    $this->articles->add($article);
  }
}
```

Abbildung 13: Foto : Code [03.12.2014, Jens Beyer]

5. Projektarbeit-Planung

Die Planung für die Umsetzung ist ein fließendes Konzept bei der Projektarbeit gewesen. Die Vorstellung von einem Ablauf wird hier dargestellt. Der erste Entwurf umfasst nicht den Projektverlauf. Dieser wird im nach Hinein aufgebrochen und erläutert.

5.1 Erster Entwurf

Controller Funktion mit Datenbankzugriffen geschätzt 10 Stunden

mit Tests 15 Stunden.

PortalData Funktionen mit Datenbankzugriffen und Logik geschätzt 15 Stunden

mit Tests 20 Stunden.

Bereitstellung der Frontendansicht geschätzt 5 Stunden

Damit würde sich ein Gesamt Aufwand von 40 Stunden ergeben.

5.2 Tatsächlicher Verlauf

Beim tatsächlichen Verlauf werden die optionalen Aufwände mit angezeigt, aber nicht mit in die Berechnung der Stunden einbezogen und sind kursiv gekennzeichnet.

Die Kombination von Controller Datenbankzugriffe und Frontend sind so eng gebunden, dass sie fast alle gleichzeitig mit einfließen.

Programmierung	
Initialisierung vom Sonata Bundle als optionales Backend Werkzeug	*2,6h 17.10.14*
Sonata Bundle DataTransformer für Select2 Integration	*2,9h 20.10.14*
Symfony Form Fields, templating umständlich, Trennung zwischen update und show	*5,0h 27.10.14*
Behandlungen von Logik in Bezug auf Datenbankzugriffe, Javascript zu flexibel und komplex, daher wird „seperation of concerns“ angewendet	*4,0h 28.10.14*
Service Injection für Facade Pattern, Entitäten Erstellung mit Abhängigkeiten. Erstellen von TagsController, ArticleController und ClientController mit Mapping von Actions auf URLs	3,5h 29.10.14
Assertion von unique Feldern mit Doctrine dauerte lange, da PHP Code mit Datenbankattributen vermischt wird und die Fehlersuche nicht leicht ist. Es folgte Umstrukturierung von PHP Entitäten Funktionen und Doctrine Struktur.	4,2h 30.10.14
TagController und ArticleController bearbeitet	5,5h 31.10.14
TagController und ClientController fertig gestellt, Doctrine 1:n und n:m Verbindung realisiert	6,5h 03.11.14
Fixtures Doctrine, PortalData class introduced, Portal DB Abfragen und view	5,1h 04.11.14

Facade Erweiterung, Portal Data Class Fertigstellung	3,5h 05.11.14
PortalData Tests mit Testsystem Erweiterung	7,3h 06.11.14
TestSysteme mit --env=test Umgebung und mysqlite Debug data von PortalData abkapseln und in PortalDataTest verankern	1,0h 07.11.14
	36,6h gesamt
Dokumentation	
Dokument erstellen in Open Office	3,3h 07.10.14
Dokumentation Ist- und Soll-Zustand	4,5h 27.11.14
Dokumentation Projektbeschneidung	2,7h 28.11.14
Dokumentation Kerninhalte	1,9h 29.11.14
Dokumentation Planung	3,1h 27.11.14
Dokumentation Komponenten	7,1h 27.11.14
Dokumentation Tests, Projektabschluss	5,2h 27.11.14
Dokumentation Anhang und Verbesserungen	4,5h 27.11.14
	32,3h gesamt

6. Komponenten und Schnittstellen

6.1 Ablauf einer Webseiten Anfrage vom Kunden

Hier wird der Ablauf von einem Besucher auf der Webseite dargestellt und wie er verlaufen soll.

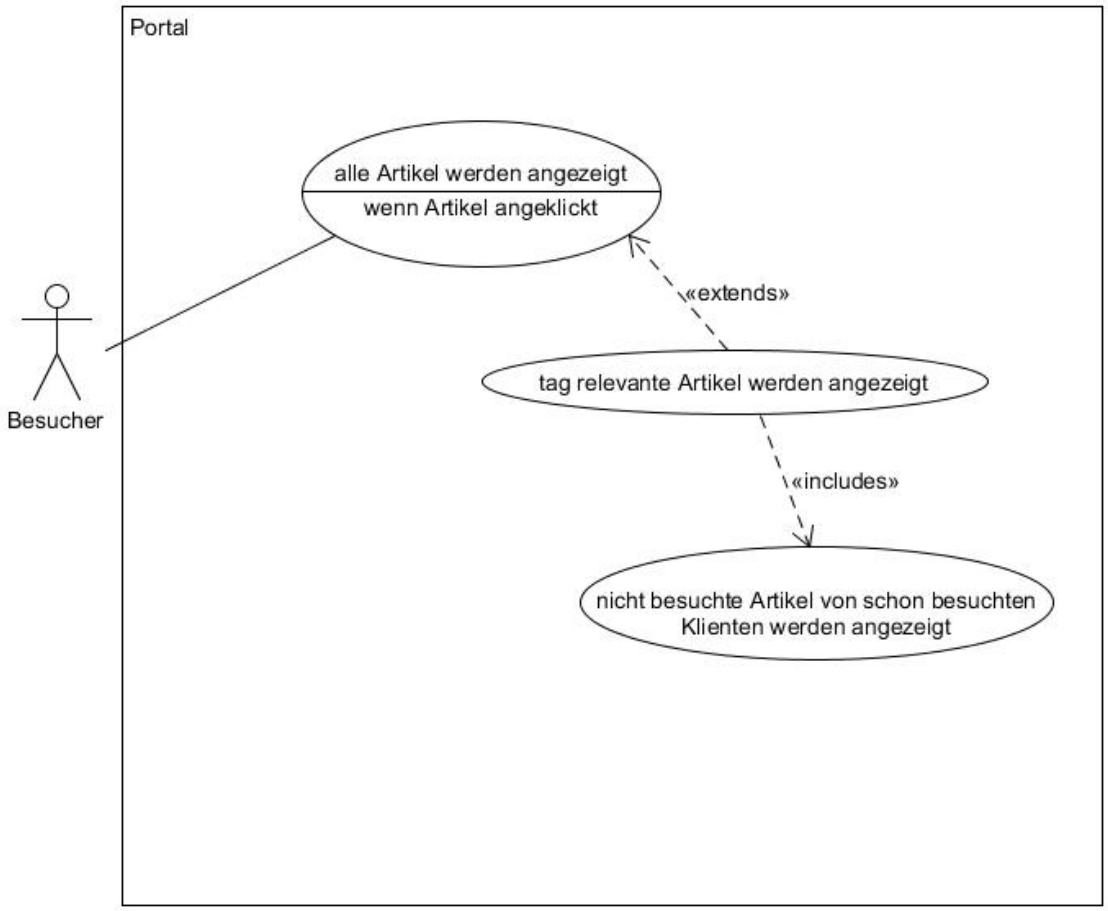


Abbildung 14: Foto :WebseitenAnfrage.jpg [27.11.2014, Jens Beyer]

6.2 Ablauf einer Anfrage in Symfony vereinfacht

Hier wird eine Webseiten Anfrage in Bezug auf die Verarbeitung im Symfony Framework dargestellt.

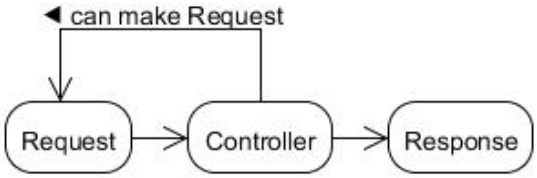


Abbildung 15: Foto :SymfonyRequest.png [02.12.2014, Jens Beyer]

6.3 Controller für create, delete oder edit Funktionen

Die benannten Controller Funktionen um Daten mit Doctrine zu pflegen werden hier aufgezeigt.

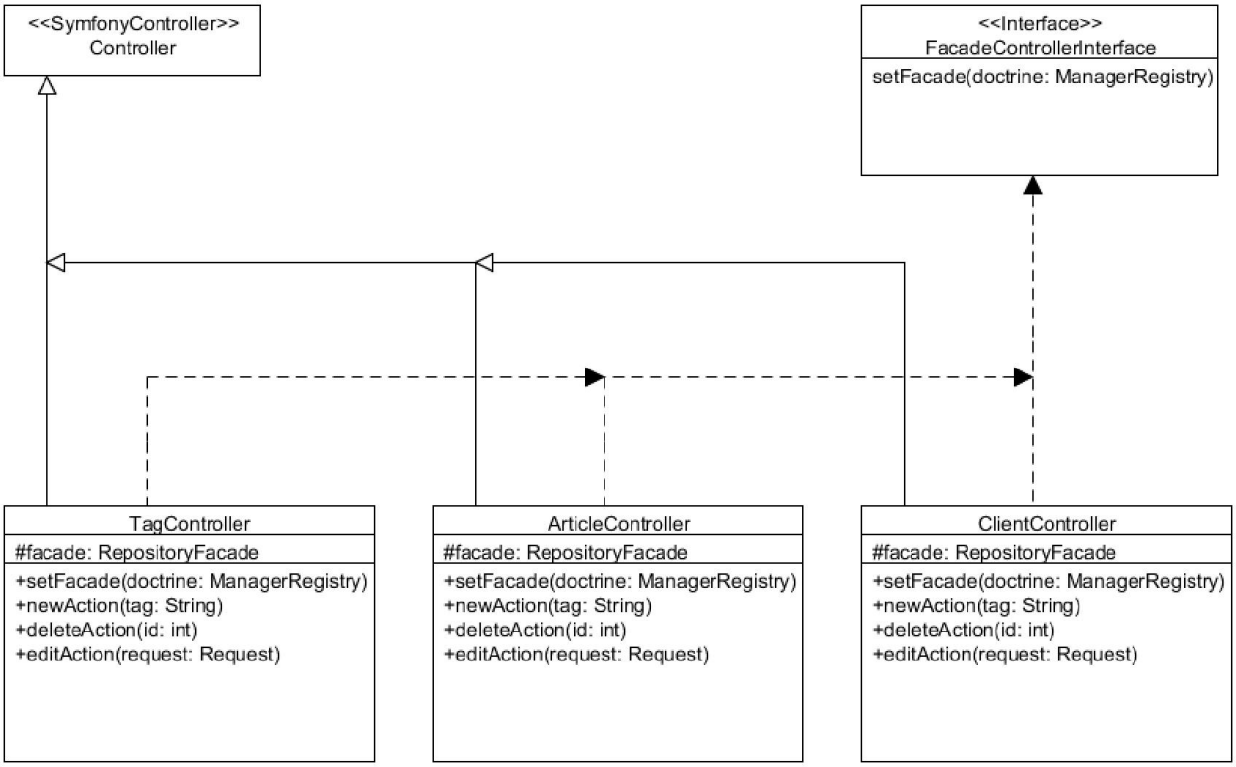


Abbildung 16: Foto : ControllerUpdateDeleteEdit.png [02.12.2014, Jens Beyer]

6.4 Controller und PortalData Interaktion

Der PortalController besitzt ein portalData Attribut welches initialisiert und bei Besuch auf einen Artikel mit der Methode visitAction aufgerufen wird.

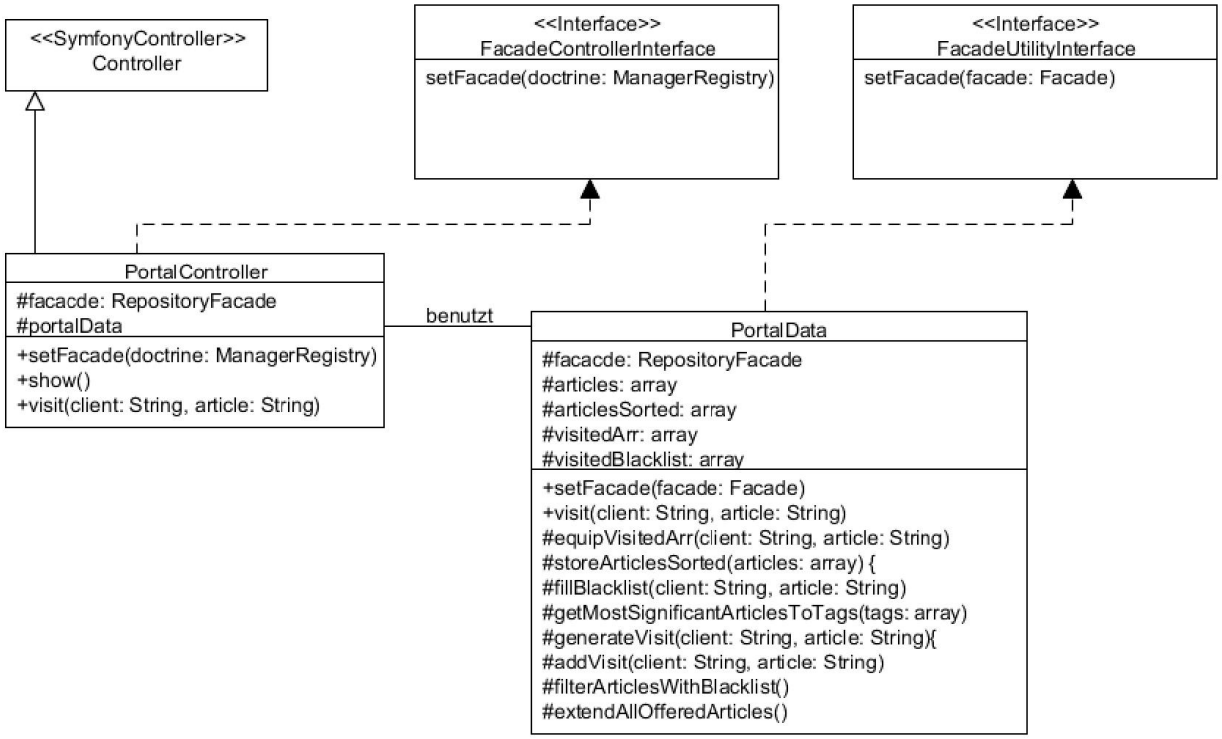


Abbildung 17: Foto : PortalController.png [02.12.2014, Jens Beyer]

Das Besuchen von Artikeln wird in einer Session gespeichert und mit der Klasse PortalData durch eine visitedBlacklist gefiltert. Besuchte Artikel sollen nicht mehr auftauchen und der Benutzer wird wie bei einem gerichteten Graphen durch die Artikel geleitet.

Der Ablauf der Controller und PortalData Klassen wird vereinfacht in einem Sequenzdiagramm dargestellt. Hier wird das FacadeRepository hinzugezogen, welches die Datenbankanfragen ansteuert.

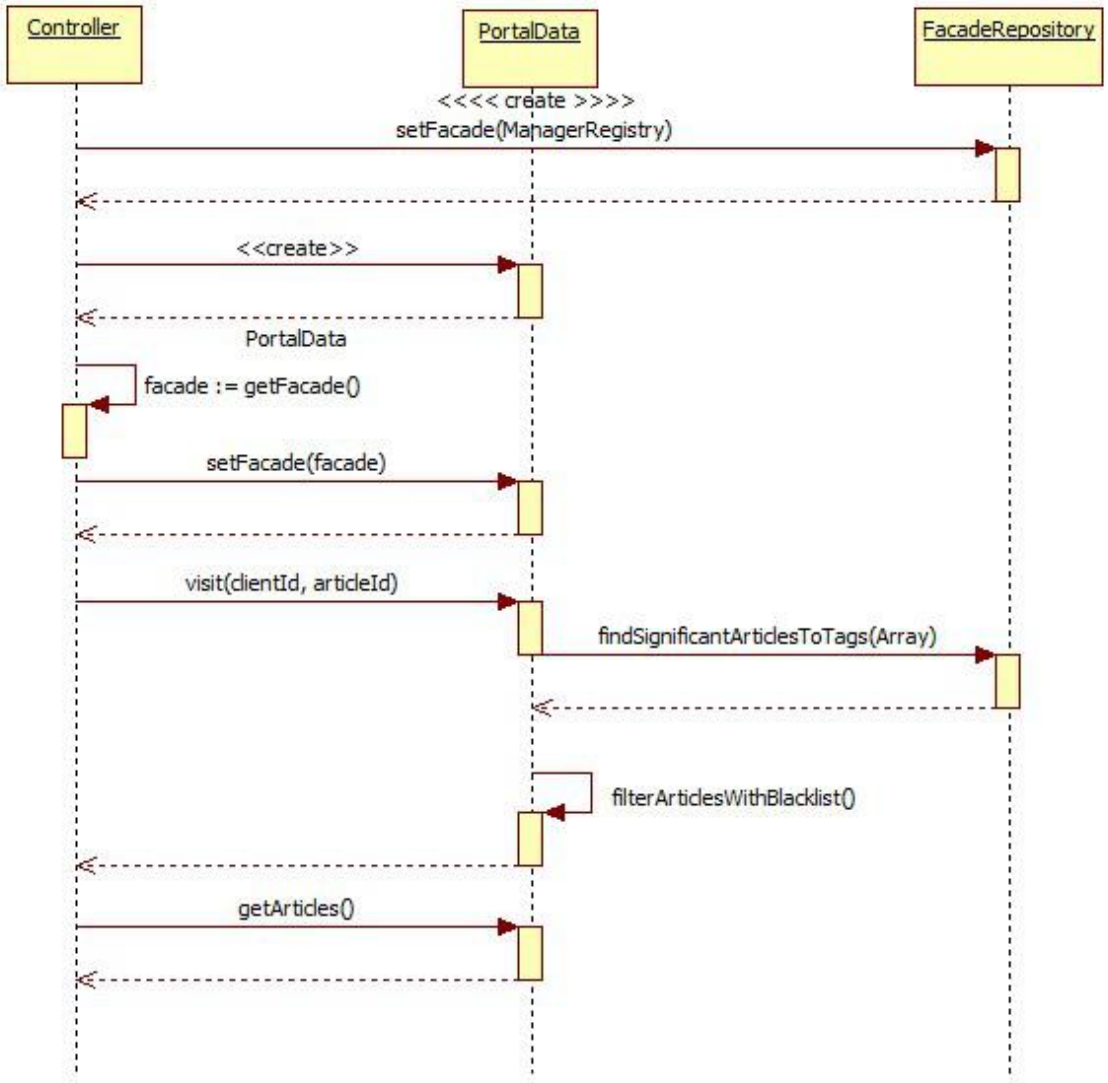


Abbildung 18: Foto : ControllerUtilitySequenz.jpg [27.11.2014, Jens Beyer]

7. Tests

7.1 Visuelle Tests

Unter den visuellen Tests werden durch eine gewisse Reihenfolge von Klicks auf Links ein sichtbares Ergebnis validiert. Diese Tests wurden bei der ersten Entwicklung von der PortalData Logik und in erster Linie der Backendpflege angewendet. Da dies der erste intensive Kontakt mit dem Symfony Framework ist, bietet sich diese Testweise an. Zumal die Validierung keiner komplexen Logik als Basis untersteht.

7.2 Funktionale Tests

Unter funktionelle Tests gehören komplexere Validierungsvorgänge, die eine Testbasis aufweisen müssen. Um die Tests bei jedem Entwickler mit den gleichen Voraussetzungen zu benutzen, wurden Konfigurationen für die „test config“ unternommen. Dazu gehört eine SQLite-Datenbank, die eine Live Datenbank auch nicht beeinflussen würde. Bei dem Projekt wurde noch kein Test-Driven-Development benutzt, da Erweiterungen für das System in Betracht gezogen werden müssen.

7.3 Das Testfeld

Die Struktur wird durch die Relationen von Artikel zu Tags getestet.

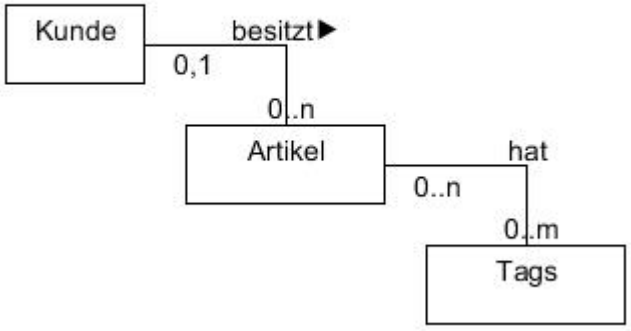


Abbildung 19: Foto :ClientArticleTags.png [02.12, Jens Beyer]

Jeder Artikel gehört einem Kunden an. Besuchte Artikel werden auf ihre Tags geprüft und weitere Artikel mit gleichen Tags werden zum besuchen offeriert. Durch eine Blackliste werden Artikel aussortiert. Durch die Abfrage von relevanten Tags des geklickten Artikels zu einem anderen Kunden B, werden eventuell neue relevante Artikel angezeigt. Es würde bedeuten, das zuvor angezeigte Artikel vom Kunden A in der offerierten Ansicht verschwinden würden. Um dem vorzubeugen werden die besuchten Kunden mit übrigen nicht besuchten Artikeln noch eingeblendet. Diese Ausnahme ist ein wichtiger Bestandteil bei der Filterung und muss getestet werden.

Die folgenden Relationen der Artikel werden hier tabellarisch dargestellt um die Abhängigkeiten zu verdeutlichen.

client	*article*	*tags*						
asv								
	stylebook		beauty	content		html	mobil	
	travelbook		beauty	content	framework	html	mobil	symfony
tdu								
	vermarkter			cms			marketing	
spiegel								
	qc	advertisement	beauty		framework		marketing	symfony

Abbildung 20: Tabelle : overviewArticles [02.12.2014, Jens Beyer]

Der Testfall von der TestKlasse PortalDataTest wird hier mit einem Aktivitätsdiagramm aufgezeigt und verdeutlicht.

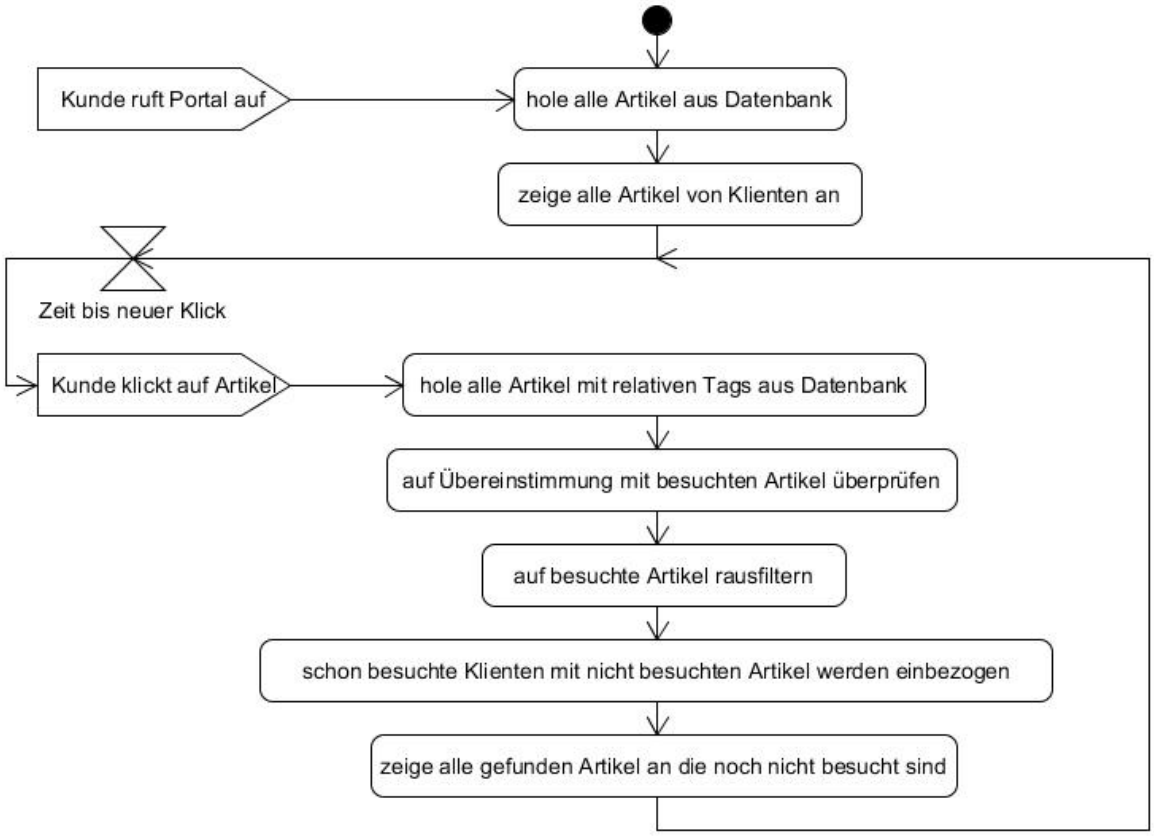


Abbildung 21: Foto :KundeArtikelAktivitaet.jpg [27.11.2014, Jens Beyer]

Folgende Testfälle bieten sich zur Validerung an.

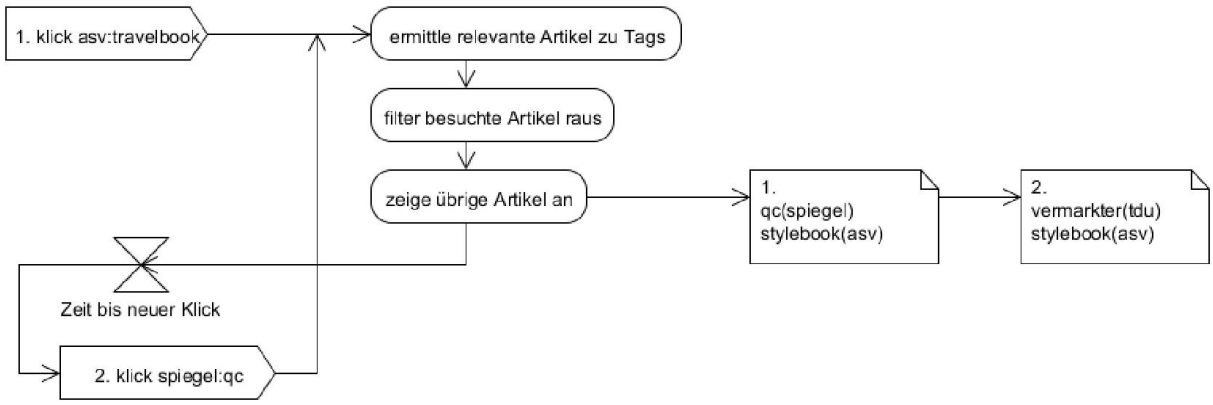


Abbildung 22: Foto : visitArticlesTest1.png [27.11.2014, Jens Beyer]

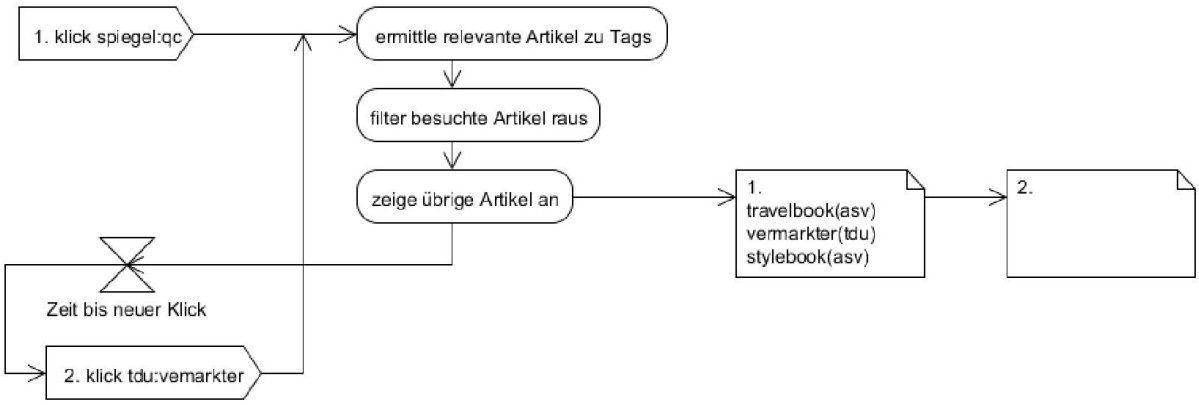


Abbildung 23: Foto : visitArticlesTest2.png [27.11.2014, Jens Beyer]

Bei diesen Tests kann man erkennen, dass offerierte Artikel wegfallen können, die vorher in Betracht gezogen wurden.

8. Projektabschluss

Das Projekt hat einen weitaus größeren Umfang und es wurde nur ein Teilaspekt betrachtet.

Durch die Umsetzung mit PHP5 konnten Funktionen verwendet werden um die Übersichtlichkeit zu erhöhen.

Doctrine ist ein umfangreiches Werkzeug, um Datenbankobjekt relationale Modelle zu entwickeln. Der Einstieg in Doctrine hat sehr viel Zeit in Anspruch genommen.

Am Ende zahlt es sich im Projekt aus, da die Erweiterungen der Tabelle weniger Aufwand erfordert.

Bei der Umsetzung wurde darauf geachtet ein testfähiges Gerüst zu wahren. Dies wurde durch Entkopplung von Symfonyobjekten realisiert.

Die Tests haben am Ende unterstützend gewirkt, um Änderungen im Backend zu testen ohne das Frontend zu benutzen. Diese Variante erwies sich als praktisch und übersichtlich. Im Nachhinein wurden die offerierten Artikel noch nach Kunden geordnet angezeigt.

Die Artikel haben nun Relationen zueinander und bilden bei Besuchen einen gerichteten Graphen,

der Endlich ist und auch dynamisch verändert werden kann, indem neue Tags oder Artikel hinzugefügt werden können.

Die Backendeinpflege wird noch mit Features benutzerfreundlicher gestaltet. Aufgrund der beschränkten Zeit fehlt Javascript Funktionalität.

Weitere Optimierungsansätze die besprochen wurden betrifft die Hinzunahme von schon besuchten Artikeln und offerierten Artikeln die nicht besucht wurden.

8.1 Änderungen zum Projektantrag

Aufgrund von Zeitlicher Einschränkung wurden die Artikel Bilder, Texte nicht mit einbezogen. Dies bezogene Erweiterungen können mit Doctrine erbracht werden und im Frontend angepasst werden.

9. Quellen Nachweis

http://symfony.com/

http://www.mysql.de/

http://ormcheatsheet.com/

http://doctrine-orm.readthedocs.org/

http://vincent.composieux.fr/article/test-your-doctrine-repository-using-a-sqlite-database

http://stackoverflow.com/questions/17091772/how-can-i-load-fixtures-from-functional-test-in-symfony-2

https://matt.drollette.com/2012/06/calling-a-method-before-every-controller-action-in-symfony2/

http://de.wikipedia.org/wiki/Klassendiagramm

10. Anhang

Im Anhang befinden sich zusätzliche Unterstützende Erklärungen und Abbildungen.

10.1 Doctrine Komponente

Die Doctrine Komponente in Symfony wird in diesem Projekt über sogenannte Annotationen konfiguriert. Annotationen sind nichts weiter als kommentierter Code, der interpretiert wird.

Folgende 3 Klassen und deren Zusammenhänge werden erläutert.

10.1.1 Client

Der Client ist eine PHP Klasse namens Client.php. Die Klasse Client wird wie folgt erstellt und einem ORM Objekt zugewiesen. Zusätzlich kann auch eine Repository Klasse hinzugefügt werden, in dem die ORM Datenbankanfragen als PHP Funktion enthalten sind.

```
/**
 * @ORM\Entity
 * @ORM\Table(name="client")
 * @ORM\Entity(repositoryClass="Acme\PortalBundle\Entity\ClientRepository")
 */
class Client
```

Abbildung: Foto : CodeClient [07.12.2014, Jens Beyer]

```
class ClientRepository extends EntityRepository

{
  public function findAllOrderedByDescription()
  {
    return $this->getEntityManager()
      ->createQuery(
        'SELECT t FROM AcmePortalBundle:Client t ORDER BY t.pos, t.name ASC'
      )
      ->getResult();
  }
}
```

Abbildung 24: Foto : CodeClientRepository [07.12.2014, Jens Beyer]

Die Relationen zu anderen Objekten sind wie folgt verankert. Hier sieht man eine OneToMany Beziehung. Dabei entspricht targetEntity(Many) dem relationalen Objekt. mappedBy ist die angelegte Entität client(One) selber. Hierbei wird auf die groß- und Kleinschreibung geachtet.

```
/**

 * @ORM\OneToMany(targetEntity="Article", mappedBy="client")
 */
protected $articles;
```

Abbildung 25: Foto : CodeClient [07.12.2014, Jens Beyer]

10.1.2 Article

Article wird wie folgt dargestellt mit Annotationen.

```
/**
 * @ORM\Entity
 * @ORM\Table(name="article")
 * @ORM\Entity(repositoryClass="Acme\PortalBundle\Entity\ArticleRepository")
 */
class Article
```

Die vorher erwähnte ManyToOne Beziehung wird hier nochmal aufgegriffen von der Article Seite zum Client. Sie enthält Informationen zur Spalte dem foreign key client_id. Neu ist hier die Relation zu Tag mit einer ManyToMany Relation.

```
/**
 * @ORM\ManyToOne(targetEntity="Client", inversedBy="articles", cascade={"persist"})
 * @ORM\JoinColumn(
 *     name="client_id",
 *     referencedColumnName="id",
 *     nullable=true
 * )
 */
protected $client;

/**
 * @ORM\ManyToMany(targetEntity="Tag", mappedBy="articles", cascade={"persist"})
 */
protected $tags;
```

Abbildung 26: Foto : CodeArticle [07.12.2014, Jens Beyer]

10.1.3 Tag

Auf der Tag Seite wird die Join Table für die ManyToMany Relation zu Article definiert. Die referenncedColumnName Attribute heißen beide id und dürfen nicht null sein.

```
/**
 * @ORM\ManyToMany(targetEntity="Article", inversedBy="name")
 * @ORM\JoinTable(
 *     name="ArticleTag",
 *     joinColumns={
 *         @ORM\JoinColumn(
 *             name="tag_id",
 *             referencedColumnName="id",
 *             nullable=false
 *         )
 *     },
 *     inverseJoinColumns={@ORM\JoinColumn(name="article_id", referencedColumnName="id",
```

```
nullable=false)}
 * )
 */
protected $articles;
```

Abbildung 27: Foto : CodeTag [07.12.2014, Jens Beyer]

10.2 PortalData komplexer Ablauf

Hier werden die Abläufe der Relationen aufgezeigt, um Artikel mit relevanten Tags zu finden und auch wieder herauszufiltern. Es fällt auf das, visitedArray und blackListArray gleiche Daten enthalten. Die Daten können sich in Zukunft ändern und abgrenzen. Daher wurden zwei Arrays genommen mit sprechenden Namen. Die Darstellung entspricht keiner Norm, sie dient nur zur Übersicht. Die Artikel werden durch angehängte Kunden z.B stylebook(asv) präsentiert.

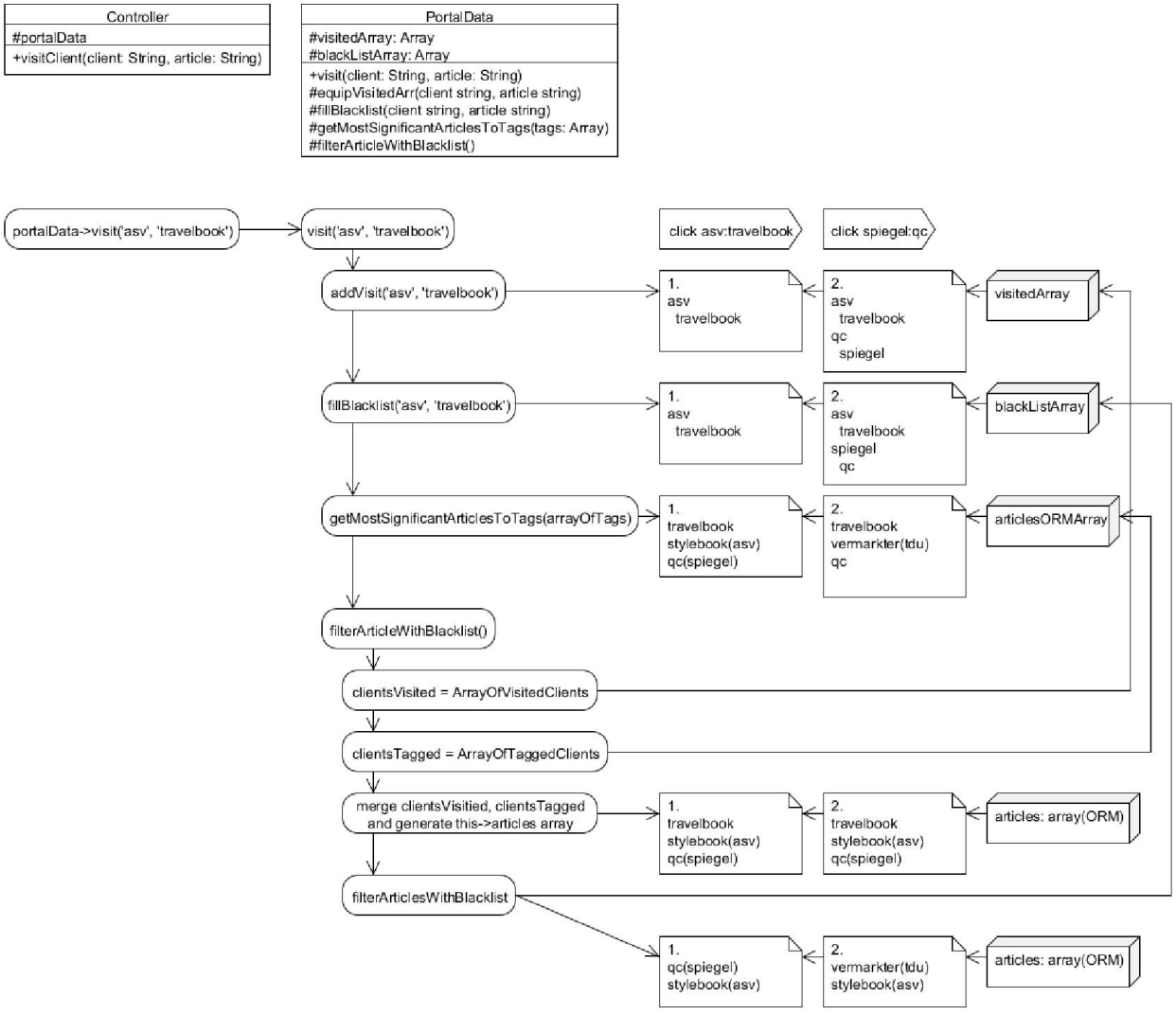


Abbildung 28: Foto : visitEplained.png [07.12.2014, Jens Beyer]

Die folgende Ansicht zeigt Artikel die während der gerichteten Suche wegfallen.

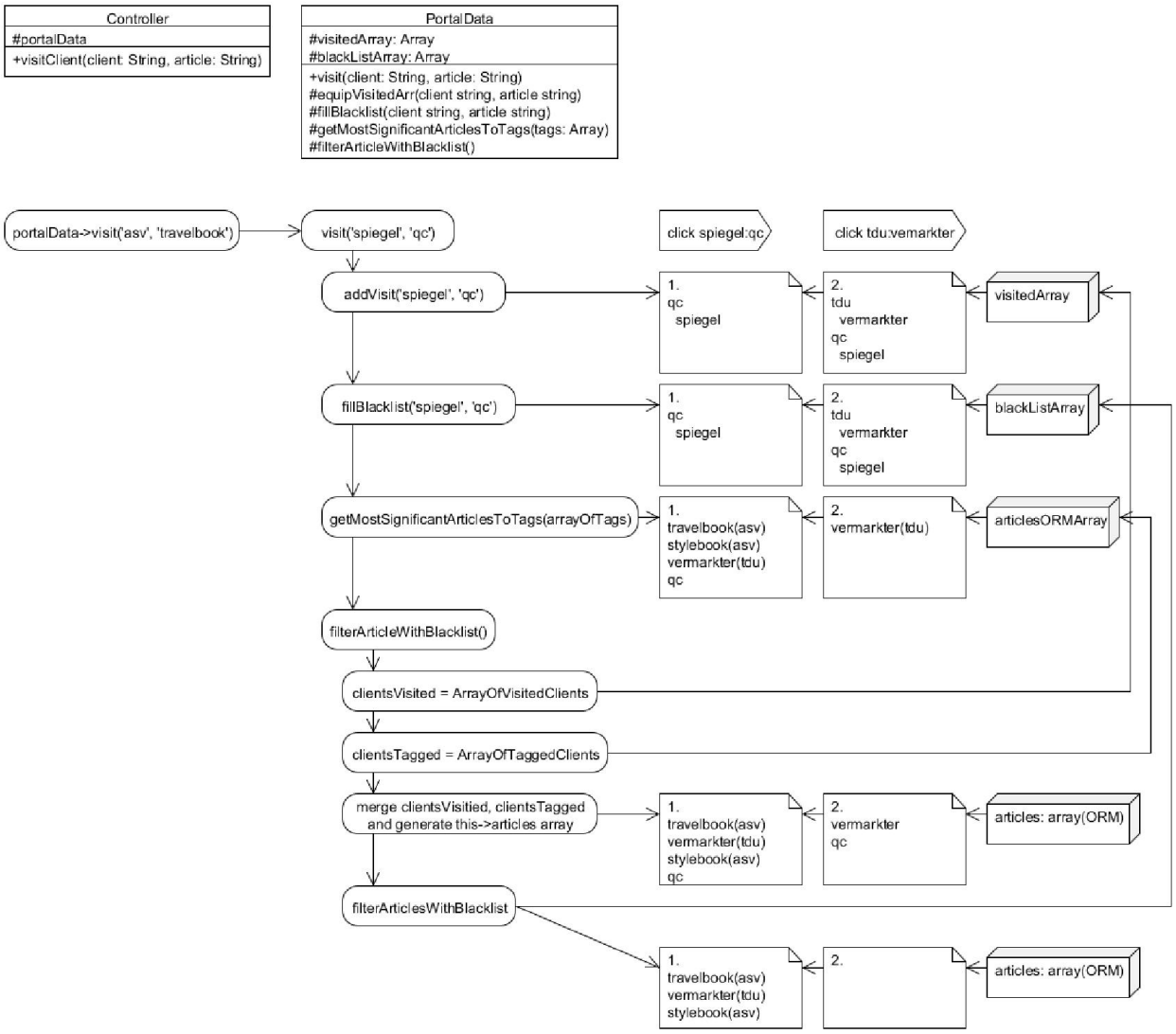


Abbildung 29: Foto : visitedExplainedLackArticle.png [07.12.2014, Jens Beyer]

Der Code der ganzen PortalData Klasse ist im folgenden zu sehen.

```php
<?php
namespace Acme\PortalBundle\Utility;
use Acme\PortalBundle\Facade\FacadeUtilityInterface;
use Doctrine\Common\Collections\ArrayCollection;
use Acme\PortalBundle\Facade\Facade;
//use \Acme\PortalBundle\Facade\FacadeInterface;
//use Acme\PortalBundle\Facade\RepositoryFacade;
//use Symfony\Bridge\Doctrine\ManagerRegistry;
use Symfony\Component\HttpFoundation\Session\Session;
class PortalData implements FacadeUtilityInterface
{
  /**
   * @var Facade
   */
  protected $facade;
  /**
   * @var Session
   */
  protected $session;
  /**
   * @var ArrayCollection
   */
  protected $articles;
  /**
   * @var Array
   */
  protected $articlesSorted;
  /**
   * @var array
   */
  protected $visitedArr;
  /**
   * @var array
   */
  protected $visitedBlacklist;

  public function __construct()
  {
    $this->articles = array();
    $this->articlesSorted = array();
    $this->visitedArr = array();
    $this->visitedBlacklist = array();
  }
  /**
   * @param Facade $facade
   * @return mixed|void
   */
  public function setFacade(Facade $facade)
  {
    $this->facade = $facade->getRepositoryFacade();
  }
```

```
public function setSession(Session $session)
{
  $this->session = $session;
}
/**
 * @param $client String
 * @param $article String
 */
public function visit($client, $article)
{
  $this->equipVisitedArr($client, $article);
  $articleDb = $this->facade->getRepository('Article')->findByDescription($article);
  if (empty($articleDb)) {
    $this->articles = $this->facade->getRepository('Article')->findAllOrderedByDescription();
    $this->storeArticlesSorted($this->articles);
    // !!!!!!!!! have to implement logging, wrong article !!!!!!!!!!!
    return;
  } else {
    $articleDb = $articleDb[0];
  }
  $tags = $articleDb->getTags();

  $this->fillBlacklist($client, $article);
  $this->articles = $this->getMostSignificantArticlesToTags($tags);
  $this->extendAllOfferedArticles(); // function is empty in this class
  $this->filterArticlesWithBlacklist();
  $this->storeArticlesSorted($this->articles);
}
/**
 * @param $client
 * @param $article
 */
protected function equipVisitedArr($client, $article)
{
  $sessionArr = $this->session->get('overview');
  if (isset($sessionArr)) {
    $this->visitedArr = $sessionArr;
    $this->addVisit($client, $article);
    $this->session->set('overview', $this->getVisitedArr());
  } else {
    $this->generateVisit($client, $article);
    $this->session->set('overview', $this->getVisitedArr());
  }
}
protected function storeArticlesSorted($articles) {
  $this->articlesSorted = array();
  foreach($articles as $article) {
    $client = $article->getClient()->getName();
    $clientPos = $article->getClient()->getPos();
    $articleName = $article->getDescription();
    if (!isset($this->articlesSorted[$clientPos])) {
      $this->articlesSorted[$clientPos] = array();
    }
```

```
    if (!isset($this->articlesSorted[$clientPos][$client])) {
      $this->articlesSorted[$clientPos][$client] = array();
    }
    $this->articlesSorted[$clientPos][$client][$articleName] = $article;
  }
  return $this->articlesSorted;
}
public function fillBlacklist($client, $article)
{
  $sessionArr = $this->session->get('blacklist');
  if (isset($sessionArr)) {
    $this->visitedBlacklist = $sessionArr;
  }
  if (!isset($this->visitedBlacklist[$client])) {
    $this->visitedBlacklist[$client] = array();
  }
  $this->visitedBlacklist[$client][$article] = $article;
  $this->session->set('blacklist', $this->visitedBlacklist);
}

/**
 * @param $tags
 * @return mixed|ArrayCollection
 */
public function getMostSsignificantArticlesToTags($tags)
{
  $tagNames = array();
  foreach($tags as $tag) {
    $tagNames[] = $tag->getName();
  }
//   $tagNames = array('marketing', 'cms');
//   $tagNames = array('marketing');
  $articlesDb = $this->facade->getRepository('Article')-
>findSsignificantArticlesToTags($tagNames);

  return $articlesDb;
//   return $articles;
}
public function generateVisit($client, $article){
  $this->visitedArr['visited'] = [];
  $this->visitedArr['visited'][$client] = [];
  $this->visitedArr['visited'][$client][$article] = $article;
}
/**
 * @param $client String
 * @param $article String
 */
public function addVisit($client, $article)
{
  if (!isset($this->visitedArr['visited'][$client])) {
    $this->visitedArr['visited'][$client] = [];
  }
  if (!isset($this->visitedArr['visited'][$client][$article])) {
```

```
    $this->visitedArr['visited'][$client][$article] = [];
  }
  $this->visitedArr['visited'][$client][$article] = $article;
}
/**
 * @return void
 */
public function filterArticlesWithBlacklist()
{
  $clientsVisited = array_keys($this->visitedArr['visited']);
  $clientsTagged = array_map(function ($article) {
    return $article->getClient()->getName();
  }, $this->articles);
  $clientsToAdd = array_merge($clientsVisited, $clientsTagged);
  $clients = $this->facade->getRepositoryFacade()->getRepository('Client')-
>findByName($clientsToAdd);
  foreach($clients as $client) {
    foreach ($client->getArticles() as $article) {
      if (!in_array($article, $this->articles)) {
        $this->articles[] = $article;
      }
    }
  }

  $this->articles = array_filter($this->articles, function ($article) {
    $clientName = $article->getClient()->getName();
    $clientsTagged[] = $clientName;
    $articleName = $article->getDescription();
    if (isset($this->visitedBlacklist[$clientName])
      && isset($this->visitedBlacklist[$clientName][$articleName])
    ) {
      $count = $article->getTags()->count();
      return false;
    }
    return true;
  });
}

protected function extendAllOfferedArticles()
{
}

/**
 * @return ArrayCollection
 */
public function getArticles()
{
  return $this->articles;
}
/**
 * @param ArrayCollection $articles
 */
public function setArticles($articles)
```

```
{
  $this->articles = $articles;
}
/**
 * @return Array
 */
public function getArticlesSorted()
{
  return $this->articlesSorted;
}
/**
 * @param Array $articlesSorted
 */
public function setArticlesSorted($articlesSorted)
{
  $this->articlesSorted = $articlesSorted;
}
/**
 * @return array
 */
public function getClientsArticles()
{
  return $this->clientsArticles;
}
/**
 * @param array $clientsArticles
 */
public function setClientsArticles($clientsArticles)
{
  $this->clientsArticles = $clientsArticles;
}

/**
 * @return array
 */
public function getVisitedBlacklist()
{
  return $this->visitedBlacklist;
}
/**
 * @param array $visitedBlacklist
 */
public function setVisitedBlacklist($visitedBlacklist)
{
  $this->visitedBlacklist = $visitedBlacklist;
}

/**
 * @return array
 */
public function getVisitedArr()
{
  return $this->visitedArr;
```

```
  }
  /**
   * @param array $visitArr
   */
  public function setVisitedArr($visitArr)
  {
    $this->visitedArr = $visitArr;
  }
}
```

Abbildung 30: Foto : CodeFürPortalData [08.12.2014, Jens Beyer]

10.3 Test Code

Der hier aufgezeigte Code ist leicht veränderbar und dient als Dokumentation für Testfälle.

```
public function testVisitWrongArticleName()
{
  $this->portalData->visit('spiegel', 'change'); // wrong article !!!!!!
  $this->result = $this->portalData->getArticlesSorted();
  // show all articles
  $this->assertTrue(isset($this->result[0]['asv']['stylebook']));
  $this->assertTrue(isset($this->result[0]['asv']['travelbook']));
  $this->assertTrue(isset($this->result[1]['tdu']['vermarkter']));
  $this->assertTrue(isset($this->result[2]['spiegel']['qc']));
}
public function testVisitSpiegel()
{
  $this->portalData->visit('spiegel', 'qc');
  $this->result = $this->portalData->getArticlesSorted();
  $this->assertTrue(isset($this->result[0]['asv']['travelbook']));
  $this->assertTrue(isset($this->result[0]['asv']['stylebook']));
  $this->assertTrue(isset($this->result[1]['tdu']['vermarkter']));
}
public function testVisitSpiegelThenVermarkter()
{
  $this->portalData->visit('spiegel', 'qc');
  $this->portalData->visit('tdu', 'vermarkter');
  $this->result = $this->portalData->getArticlesSorted();
  $this->assertTrue(empty($this->result));
}
public function testVisitAsvThenTdu()
{
  $this->portalData->visit('asv', 'travelbook');
  $this->portalData->visit('tdu', 'vermarkter');
  $this->result = $this->generateClientsArticles($this->portalData->getArticles());
  $this->assertTrue(isset($this->result[0]['asv']['stylebook']));
}
```

Abbildung 31: Foto : TestCodeFürPortalData [07.12.2014, Jens Beyer]