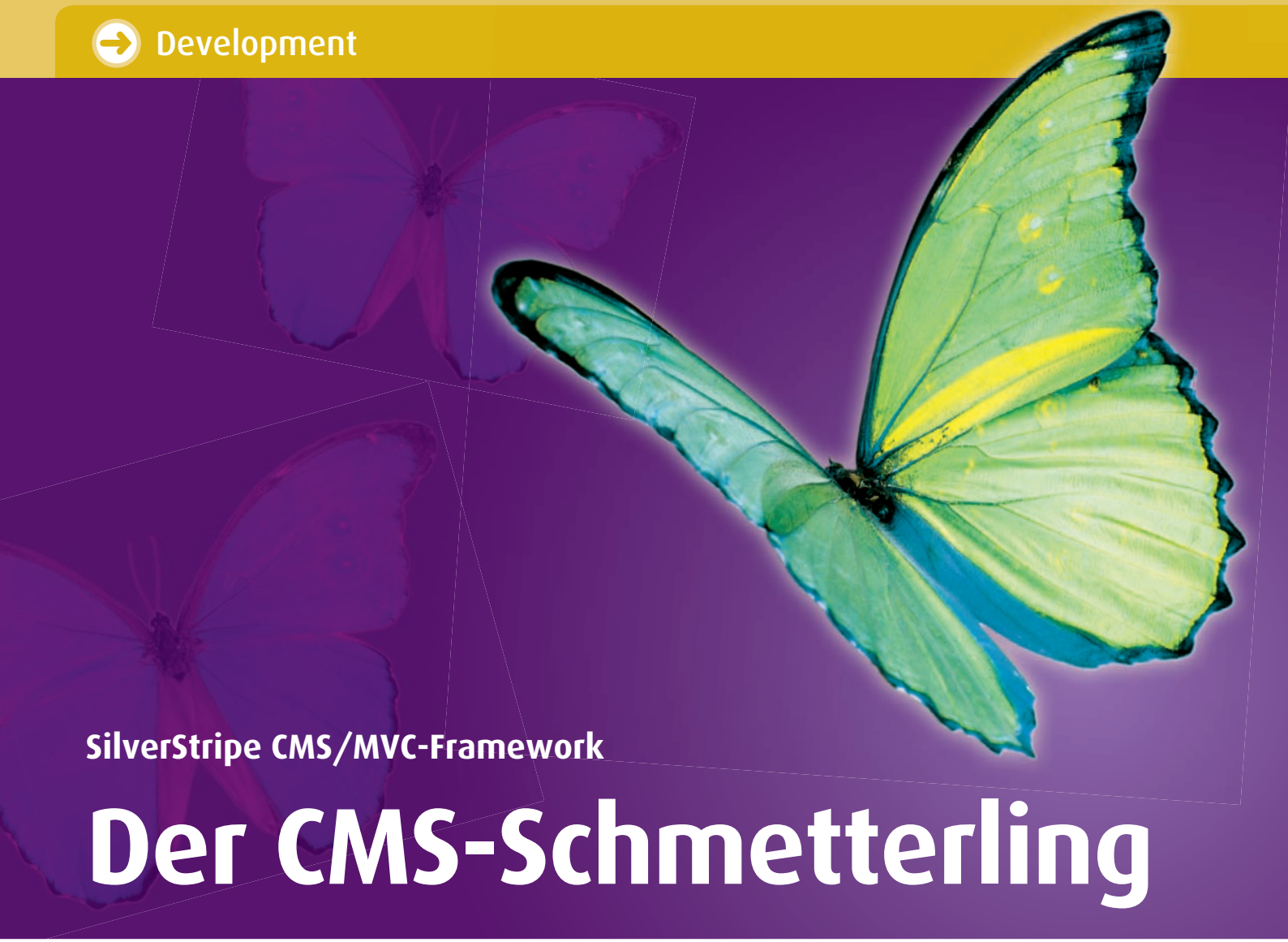

SilverStripe CMS/MVC-Framework

Der CMS-Schmetterling

Steven Broschart

In Deutschland bisweilen noch recht unbekannt, entwickelt sich – unter „Aufsicht" von Google – ein sehr interessantes Open Source CMS, das mit seinem MVC-Framework als Unterbau auf eine sehr einfache und schnelle Implementierung von Inhalten und Zusatzfunktionalitäten setzt.

Der *Large Silverstripe* (großer Silberstrich, [1]) ist eigentlich ein australischer Schmetterling, gehört zu den tropischen Vertretern seiner Art und fällt durch seine kontrastreiche Zeichnung auf. Obwohl die Entwickler des SilverStripe CMS [2] ein paar Kilometer weiter in Wellington (Neuseeland) ansässig sind, weist deren CMS durchaus einige Parallelen auf: Erst nach genauerem Hinsehen entpuppt sich ein schönes, leichtfüßiges Etwas, das gerade durch seine Unterschiede zu Anderen punkten kann.

Warum es sich also doch lohnen könnte, ein weiteres Open Source CMS des bereits aufgeteilt erscheinenden Markts zu erforschen, soll im folgenden Artikel beschrieben werden, der neben der Systembeschreibung auch noch einen kleinen Workshop beinhaltet.

Gesamtkonzept und Features

SilverStripe ist ein auf PHP und MySQL basierendes Content-Management-System mit integriertem MVC-Framework. Das CMS bietet dabei die Standard-Features, die man von einem modernen System erwarten kann. Dazu gehören unter anderem eine praxisorientierte Rechtevergabe, beziehungsweise Benutzerverwaltung, Formularwizard und Newslettersystem, Versionierung und Staging, einfaches Workflowsystem, On-the-Fly-Menügenerierung, Unterstützung von OpenID, grafisches Reportingsystem sowie eine Bilder- und Dokumentenverwaltung. Um im

Web2.0-Zeitalter dem Mitmachbedürfnis gerecht zu werden, kann zudem jede Seite mit einer Kommentarfunktion ausgestattet werden. Das Backend bietet außerdem an vielen Stellen die Möglichkeit, Inhalte per Drag-and-Drop zu verwalten.

Neben diesen für Redakteure interessanten Bereichen lohnt unbedingt auch ein ausführlicher Blick unter die Haube: Hier werkelt ein MVC-Framework, das sicherlich für Freude bei den Programmierern sorgen wird (dazu später mehr). Außerdem verrichtet hier ein auch zwingend notwendiges Caching-System seine Arbeit, damit die einmal über das Framework gerenderten Seiten auch zügig ausgeliefert werden können.

Was bisher geschah

Mit der im Februar 2007 erschienen Version 2.0 entschlossen sich die Entwickler, SilverStripe nicht länger kommerziell zu vermarkten, sondern unter der BSD Licence als Open Source CMS frei zur Verfügung zu stellen. Eine gute Entscheidung, denn Google entschied sich im alljährlich veranstalteten *Google Summer of Code* für SilverStripe als teilnehmendes Projekt. SilverStripe konnte so in relativ kurzer Zeit nicht nur weiter an Qualität, sondern vor allem auch an Bekanntheit gewinnen. Die Zusammenarbeit setzte sich anschließend mit dem ebenfalls von Google veranstalteten *Highly Open Participation Contest* fort.

Nun standen also – im Branding-Schatten von Google – ein paar Features auf dem Flipchart, die Lust auf mehr machen sollten: Einfachheit, Suchmaschinenfreundlichkeit und das MVC-Framework.

Ist „einfach" einfach besser?

Nach Aussage der Entwickler soll SilverStripe in allen Belangen „einfach" sein: Einfache Installation, einfache Backend-Bedienung, einfach zu skalieren, einfache Template-Entwicklung und – einfache Funktionserweiterung über das MVC-Framework. Da sich natürlich kein CMS-Entwickler das Feature „Schwer zu installieren" oder „Ultrakomplexe Rechteverwaltung" aufs eigene Datenblatt schreibt, sollte man solche Werbeaussagen sicherlich immer mit einem gewissen Maß an Skepsis betrachten. Auf der anderen Seite könnte „einfach" aber auch heißen, dass möglicherweise wichtige Komponenten *einfach* fehlen. Da hilft nur eines: Überprüfen ...

Nach einer nun wirklich problemlosen, schnellen und einfachen Installation erscheint SilverStripe mit dem ersten Aufruf des Backends in der Tat recht aufgeräumt – im Vergleich zu anderen CMS beinahe spartanisch. Trotzdem sorgt genau diese Übersichtlichkeit für eine intuitive Navigation zu den Inhalten der ersten Seite. Auch die Eingabemaske der

SilverStripe-Backend

Nach der Installation kann man sich mit den zuvor hinterlegten Zugangsdaten über *www.meine-domain.de/Security/login* einloggen.

- Über die Kopfnavigation lassen sich die Kernbereiche des CMS sowie Systemerweiterungen in Form von so genannten Modulen erreichen. Beim Aufruf des Backends ist Site Content aktiv.
- In dieser Ansicht erscheint auf der linken Seite der Seitenbaum, der – wie auch bei anderen Systemen – zur Verwaltung der Seitenstruktur dient. Hier können Seiten neu angelegt, gelöscht, verschoben oder aber auch durchsucht und in ihrem Status (Draft, Published) modifiziert werden.
- Modifizierte Seiten werden zunächst nur mit einer entsprechenden Markierung versehen. Änderungen werden erst nach dem Abspeichern aktiv.
- Nach der Auswahl einer gewünschten Seite im Seitenbaum können die Inhalte über den Eingabebereich verwaltet werden.
- SilverStripe verfügt über ein Stagingsystem, das erlaubt, eine Seite vor der Onlinestellung in einem von außen nicht sichtbaren Bereich als Entwurf abzulegen (Draft). Über die *Page View*-Navigation kann man sich die aktuell hinterlegte Entwurfsversion, aber auch das bereits publizierte Dokument ansehen.
- Das gewählte Dokument kann publiziert oder als Entwurf (Draft) abgespeichert werden. Alternativ kann man über *Unpublish* die aktuelle Seite aus dem öffentlich zugänglichen Bereich entfernen.

Es gibt übrigens keinen üblichen Setup- oder Konfigurationsbereich, wie man das von anderen Systemen her kennt.

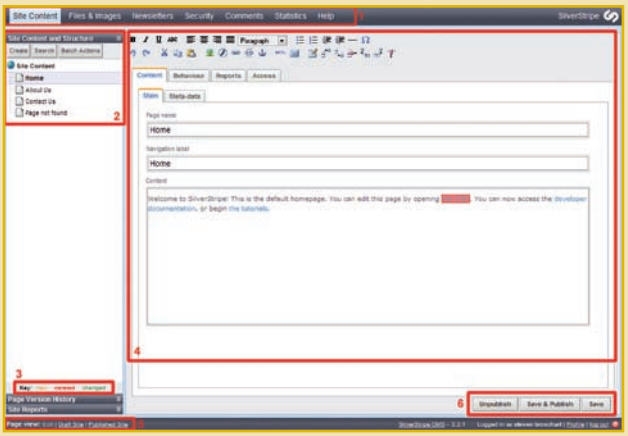

Abb. 1: Das SilverStripe Backend

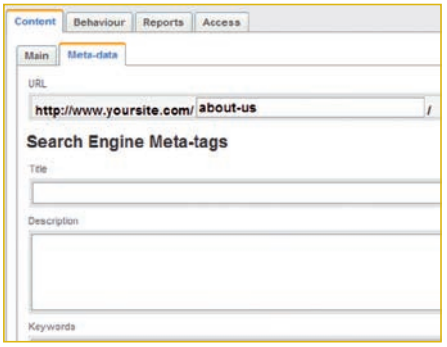

Abb. 2: Verwaltung der Metadaten

Inhalte dieser ersten Seite erscheint recht aufgeräumt und lässt vermuten, hier recht schnell an seine Grenzen zu stoßen, wenn es um weitere, individuelle Eingabefelder oder die Ergänzung weiterer Funktionalitäten geht.

SilverStripe konzentriert sich hier auf die wesentlichen Komponenten, die für viele Anwendungsfälle bereits unverändert ausreichen sollten. Wie noch im weiteren Verlauf des Artikels gezeigt wird, kann man mit ein wenig PHP-Erfahrung nun aber trotzdem sehr schnell zusätzliche Eingabefelder oder weitere Funktionalität implementieren – und zwar schneller, als dies mit einem anderen, nicht MVC-basierten System möglich wäre.

Das System stellt damit out-of-the-box alles bereit, was als Basis für eine neue Webanwendung genutzt und dann weiter ausgebaut werden kann. Die einfachen Erweiterungsmöglichkeiten kompensieren die Tatsache, dass SilverStripe bisher noch von einer recht überschaubaren Entwicklergemeinde getragen wird und deshalb auch nur wenige fertige Erweiterungen (bei SilverStripe *Module* genannt) existieren. Dass die Entwicklung im MVC-Framework nebenbei auch noch Spaß macht, motiviert zur Programmierung und garantiert damit eine große Bereitschaft zur Entwicklung weiterer Module durch die stetig wachsende Community.

Seid nett – auch zu Suchmaschinen!

Gerade im Hinblick auf die Zusammenarbeit mit Google könnte man beim Stichwort *Suchmaschinenfreundlichkeit* auf dumme Gedanken kommen: Ist da die SEO etwa schon ab Werk eingebaut?

Aber mitnichten: Es geht alles mit rechten Dingen zu. SilverStripe stellt lediglich die Verwaltung selbstsprechender URLs und weiterer Metainformationen zur Verfügung (über den Metadata-Tab, siehe Abbildung 2), unterstützt zudem die automatische Generierung von *sitemap.xml*-Dateien und macht auf nicht-validen XHTML-Code aufmerksam.

Damit schafft das System die Grundvoraussetzungen für eine saubere Indexierung.

Wie der geübte SEOler weiß, ist für ein gutes Ranking allerdings noch wesentlich mehr notwendig. Die Abstimmung von Titeln, Überschriften, Fließtext und Verlinkungen ist entscheidend für eine gute Platzierung in den Suchergebnissen. Diese Arbeit kann und soll dem Redakteur von keinem CMS abgenommen werden.

MVC-Framework

Das wirklich besondere an SilverStripe ist der bereits erwähnte MVC-Unterbau, der, gerade bei der Suche nach einem neuen System, von entscheidender Bedeutung sein kann. Wer sich bereits mit Ruby on Rails und dem dort ebenfalls verwendeten MVC Design Pattern beschäftigt hat, wird auch hier Altbekanntes wiederfinden und sich schnell zu Hause fühlen.

CMS und MVC arbeiten bei SilverStripe Hand in Hand: Während das CMS über das Backend angesprochen und primär zum Verwalten von Inhalten verwendet wird, kümmert sich das Framework darum, diese Inhalte für die Ausgabe vorzubereiten und auf Nutzereingaben zu reagieren.

Bevor wir mit dem Workshop beginnen, sollten Sie für eine erste Orientierung einen Blick auf die Ordnerstruktur werfen:

- *assets* (Ablageordner für über das CMS hochgeladene Dateien)

SilverStripe-Entwickler: In Neuseeland und weltweit

SilverStripe-Mitbegründer und CMO Sigurd Magnusson freut sich über eine stetig wachsende Community und auch über weitere pfiffige Programmierer aus der ganzen Welt, die sich in das Projekt der neuseeländischen Firma einbringen möchten.

Das junge Kernteam besteht in Wellington aus insgesamt 16 Mitarbeitern, zu denen auch der Deutsche Ingo Schommer gehört. Der Senior Developer hat vor knapp anderthalb Jahren nicht nur Deutschland, sondern auch TYPO3 den Rücken gekehrt und fühlt sich nach eigenen Angaben mit dieser Entscheidung sehr wohl. Bei TYPO3 empfindet er vor allem die zeitintensiven und unübersichtlichen Konfigurationsarbeiten als Hemmnis, während er bei SilverStripe wesentlich direkter und produktiver zum Ziel gelange.

Weitere Informationen, das Forum sowie alle benötigten Installationsfiles stehen unter www.silverstripe.com bereit.

Technische Voraussetzungen

SilverStripe erfordert PHP mindestens in der Version 5.04, empfohlen wird jedoch 5.2 und höher sowie mindestens 32MB zugesicherter Speicher. Als Datenbank kommt für die zu Redaktionsschluss erhältliche Version 2.2.1 nur MySQL (mindestens Version 4.1) in Frage. Die Unterstützung für PostgreSQL, SQLite und MS SQL ist aber in Vorbereitung. Als Webserver empfehlen die Entwickler den Apache oder den Lighttpd.

Google Summer of Code

Seit dem Jahr 2005 richtet Google ein alljährliches Programmierstipendium aus, bei dem sich Studenten aus aller Welt bewerben und an einem Open-Source-Projekt mitentwickeln können. Beim Erreichen der von den Entwicklern gesteckten Ziele wird insgesamt eine Prämie von 5000 US-Dollar ausgeschüttet.

Installation

Die Installation des Systems geht einfach von der Hand: Hierfür steht entweder ein Windows Installer mit allen erforderlichen Programmen bereit (Lighttpd, PHP5, MySQL) oder klassisch die Archivdatei zum Entpacken auf einem bereits vorbereiteten Serversystem.

Bei der Installation über den klassischen Weg wird der Installer nach dem Upload aller erforderlichen Dateien über *www.meine-domain.de* aufgerufen. Neben den Informationen zum Administrator und zur Datenbank möchte dieser wissen, ob später ein leeres oder das Demo-Template verwendet werden soll. Für das Tutorial in diesem Artikel sollte die Standardeinstellung (BlackCandy) beibehalten werden.

Sind soweit alle Daten eingegeben und gibt auch der Requirements-Check grünes Licht, kann über Betätigen des Install-SilverStripe-Buttons die Installation begonnen werden. Dieser Vorgang benötigt nur wenige Augenblicke.

Abschließend kann noch die Installationsdatei mit einem Klick gelöscht werden. Praktisch.

Zur Sicherheit sollte man auch noch den letzten Vorschlag des Installers beachten: Schreibrechte sollten nur für den Ordner */assets/* gewährt werden. Fertig.

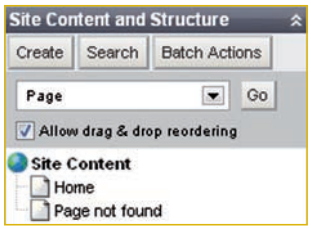

Abb. 3: Verwaltung über den Seitenbaum

- *auth_openid* (OpenID-Modul)
- *cms* (CMS Backend)
- *jsparty* (AJAX Files)
- *mysite* (alle Daten für die eigene Seite)
- *sapphire* (Framework)
- *themes* (Demo-Templates, Ordnerstruktur wie unter *mysite*)

Für den folgenden Workshop sind lediglich die Ordner *mysite* und *themes* relevant. */mysite/code* enthält die Daten, die für Model- und Controller-Schicht notwendig sind. Unter */themes/blackcandy/templates/* sind alle Daten für die View-Schicht abgelegt.

Workshop Teil I: CMS – Seitenstruktur und Inhalte

Für diesen Praxisteil soll eine einfache Internetpräsenz für ein fiktives Atelier erstellt werden. Dafür gehen wir von folgender Seitenstruktur aus:

- Home
- Über uns
- Produktkatalog
- Kontakt

Wir können also aus der Basisinstallation zunächst einmal die Seiten „About Us“ und „Contact Us“ löschen. Dazu müssen Sie oberhalb des Seitenbaums den Button *Batch Actions* betätigen. Nun können die gewünschten Seiten über die Checkbox markiert und anschließend über *Delete the selected pages* gelöscht werden. Ist die gewählte Seite bereits publiziert, wird diese erst dann gelöscht, wenn sie über *Unpublish* deaktiviert wird. Bis dahin erscheint sie noch durchgestrichen in Rot. Nun können über den Button *Create* neue Seiten angelegt werden.

Mit einem Klick auf *Go* wird nun eine neue Seite des Typs *Page* angelegt. Für ein einfaches Arrangieren der Seiten via Drag-and-Drop markieren Sie am besten auch noch gleichnamige Checkbox.

Anschließend findet sich im Seitenbaum eine neue Seite mit dem Namen „New Page“ wieder. Die orange Farbe deutet darauf hin, dass es sich hier um eine neue, noch nicht publizierte Seite handelt.

Hinweis: Eine neue Seite wird hierarchisch immer unterhalb des zuvor ausgewählten Elementes eingefügt. Damit also „New Page“ mit den anderen Seiten auf gleicher Ebene erscheint, müssen Sie vor dem Erstellen das Wurzelelement *Site Content* ausgewählt haben. Bei Bedarf lässt sich die neue Seite nun auch über Drag-and-Drop verschieben.

Zur inhaltlichen Bearbeitung kann die Seite jetzt über den Seitenbaum ausgewählt und anschließend über den *Content*-Tab editiert werden. Da diese Seite nun zu „Über uns“ werden soll, wird unter *Page Name* auch „Über uns“ eingetragen. Auch in der Navigation soll dies später auftauchen, weshalb wir das Gleiche nochmal unter *Navigation Label* eintragen. Das Feld *Content* nimmt nun den eigentlichen Inhalt der Seite auf und kann mit beliebigem Text bestückt werden, was wir an dieser Stelle gerne Ihrer Kreativität überlassen möchten. Abschließend können die Eingaben über den *Save & Publish*-Button unten rechts gespeichert und veröffentlicht werden. Der zuvor orangefarbene Eintrag im Seitenbaum erscheint nun in Schwarz und markiert „Über uns“ damit als publiziert.

> Kritiker empfinden das Schichtenkonzept als zu starr und unflexibel

Nach dieser „normalen“ Seite soll nun die Kontaktseite mit einem Kontaktformular erstellt werden. Dies geht ebenfalls wieder über den eben beschriebenen Weg. Als Seitentyp wählen wir aus dem Dropdown-Menü anstelle von *Page* nun *Contact Form* aus und klicken auf *Go*. Die neue Seite mit dem Titel „New UserDefinedForm“ sollte wie oben beschrieben mit der Bezeichnung „Kontakt“ versehen werden. An dieser Stelle können Sie übrigens über das *Content*-Eingabefeld auch noch einen Text hinterlegen, der mit dem späteren Formular angezeigt werden soll (beispielsweise das Impressum). Jetzt kann über den *Form*-Tab nun das eigentliche Formular zusammengestellt werden.

Die gewünschten Eingabefelder können jetzt über die Toolbar im Kopf- oder Fußbereich hinzugefügt werden. In unserem Beispiel handelt es sich um Text- und E-Mail-Eingabefelder. Der Platzhalter (*Enter Question*) muss anschließend noch mit der gewünschten Feldbeschreibung ausgetauscht werden. Weitere Detailkonfigurationen können über das Icon rechts vom Eingabefeld vorgenommen werden. Außerdem lassen sich alle Felder per Drag-and-Drop neu sortieren. Abschließend müssen wir noch das Ziel einer Formularanfrage eintragen. Dazu geben Sie unter *Email submission* to einfach die gewünschte Zieladresse an. Alle Anfragen werden übrigens gespeichert und können später über den *Submission*-Tab eingesehen werden. Sollte die Standardbeschriftung des *Submit*-Buttons nicht gefallen, lässt sich diese noch über den letzten Eintrag, über *Text on submit button* beliebig ändern.

Nach einem finalen Klick auf *Save & Publish* kann das erstellte Formular nun endlich auch online getestet werden. Möchten Sie die Bestätigungsnachricht, die nach dem Absenden des Formulars erscheint, noch ändern, können Sie dies über den *On complete*-Tab (unterhalb vom *Content*-Tab) nachholen. Befindet sich die Kontaktseite innerhalb der Navigation noch nicht an der richtigen Stelle, können Sie das wie beschrieben über den Seitenbaum per Drag-and-Drop ganz einfach korrigieren.

Bevor wir nun im zweiten Teil des Workshops etwas in die Tiefe gehen, steht es Ihnen natürlich noch frei, die Inhalte der verbliebenen Seite „Home“ an persönliche Gegebenheiten anzupassen.

Workshop Teil II: MVC – Der Produktkatalog

Wie schon aus der oben aufgezeigten Seitenstruktur ersichtlich, soll nun noch ein Produktkatalog in das System integriert werden. Dieser soll insgesamt aus drei unterschiedlichen Kategorien bestehen:

Plastiken, Gemälde, Fotografien. Im CMS legen wir dafür eine Seite mit dem Namen „Produktkatalog“ an. Diese Seite soll nun als „Rahmen“ für die genannten Kategorien herhalten. Im Ordner */mysite/code* wird deshalb eine erste eigene Datei mit dem Namen *Categories.php* angelegt (Groß- und Kleinschreibung bitte unbedingt beachten), die folgenden Code enthält:

```
<?php
class Categories extends Page {
  static $allowed_children = array('Category');
}
class Categories_Controller extends Page_Controller {}
?>
```

Dieser Code definiert einen neuen Seitentyp mit dem Namen *Categories*. Die Klasse *Categories* basiert dabei auf dem Standard-Seitentyp *Page* und gibt vor, dass nur Seiten des Typs *Category* unterhalb angeordnet werden dürfen. Diese Klasse stellt übrigens unsere erste Anweisung der Model-Schicht dar.

Die Klasse *Categories_Controller* definiert die Anweisungen für die Controller-Schicht. Da es hier nur um einen Seitentyp für einen „Rahmen“ ohne weitere Funktionalitäten geht, genügt es, wenn sie die Eigenschaften von *Page_Controller* erbt und auf weiteren Code verzichtet.

Hinweis: Zur Definition einer Model- und Controller-Klasse muss auf spezielle Namenskonventionen geachtet werden. Die Klasse für die Model-Schicht ist dabei gleichlautend mit dem Dateinamen. Die Controller-Klasse enthält ebenfalls den Dateinamen, dem allerdings noch ein *_Controller* angefügt wird. In den meisten Fällen erbt die Model- und Controller-Schicht dabei von *Page*, damit auch neue Seiten direkt auf die Funktionalität der Standardseite zurückgreifen können.

Damit das Framework die Datenbankstruktur entsprechend unserer Änderungen aktualisiert und den Template-Cache löscht, muss – als Administrator angemeldet – die folgende URL aufgerufen werden:

http://www.meine-domain.de/db/build?flush=1

Diesen Refresh-Befehl sollten Sie sich merken, da er zur Aktualisierung nach jeder Änderung an der Datenstruktur aufgerufen werden muss. Damit haben Sie soeben Ihren ersten neuen Seitentyp definiert. Der Seitentyp von „Produktkatalog“ kann nun über den *Behaviour*-Tab auf den neuen Typ *Categories* umgestellt werden.

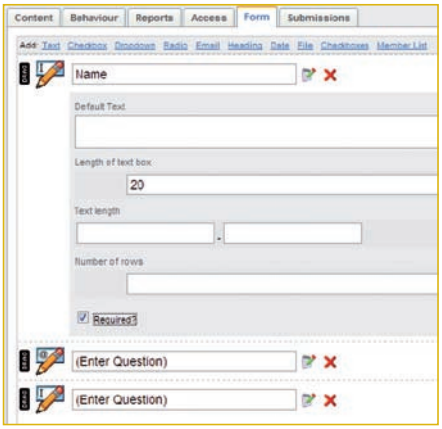

Abb. 4: Formular-Wizard

Bevor wir nun die drei Seiten für die Kategorien anlegen, wollen wir zunächst den zugehörigen Code definieren. Ziel soll sein, Produkte einer zentralen Liste über das Backend frei auf die drei Kategorien verteilen und verwalten zu können. Jedes Produkt soll dabei über einen Titel, einen Beschreibungstext und ein Bild spezifiziert werden können. Dazu müssen wir zunächst unter */mysite/code* eine weitere Datei anlegen, die wir am sinnvollsten *Category.php* nennen. Mit dieser Datei definieren wir einen weiteren Seitentyp, nämlich *Category*. Außerdem wird hier definiert, dass wir im CMS Backend einen weiteren Tab mit den Produktdaten in Form einer editierbaren Liste wünschen, siehe Listing 1.

Über *static $has_many* ... wird eine erste Abhängigkeit für die Datenbank festgelegt, die besagt, dass *Category* viele *Products* aufnehmen kann. *MyProduct* ist dabei der von uns definierte Name der Abhängigkeit (brauchen wir später noch für die Ausgabe), *Product* der Klassenname.

Es folgt im weiteren Verlauf das Überschreiben einer Framework-internen Funktion: Innerhalb von *getCMSFields()* werden zunächst die Eingabefelder als Objekt ausgelesen. Über die Instanzierung von *HasManyComplexTableField* wird eine Liste für

die Produkte mit den Spalten „Productname“ und „Description“ definiert. Abschließend wird diese über *addFieldToTab* in das neue Tab *Products* (adressierbar über *Root.Content.Products*) integriert. Tabs werden auf diese Art automatisch angelegt, sollten sie noch nicht existieren.

Jetzt fehlt nur noch die Definition des Produkts selbst: Hierzu müssen wir eine weitere Datei unter */mysite/code* anlegen, die wir am besten *Product.php* nennen und mit dem Code aus Listing 2 bestücken.

Diese Datei definiert über *static $db* ... die Datenbankspalten, die wir für unsere Produktdaten benötigen. Über *static $has_one* ... folgt eine weitere Abhängigkeitsdefinition, die für jedes Produkt nur eine Kategorie und ein Bild zulässt. Die Funktion *getCMSFields_forPopup()* verhält sich ähnlich wie oben beschrieben. Hier wird allerdings ein Eingabeformular für die Ausgabe in einem Pop-up vorbereitet.

Nach erneutem Refresh des Frameworks können wir nun die drei weiteren Seiten für die Kategorien anlegen. Dazu wählen Sie zunächst die Seite „Produktkatalog“ aus dem Seitenbaum an und klicken anschließend auf *Create*. In der Drop-down-Liste für den Seitentyp sollte nun direkt *Category* erscheinen, da sie an dieser Stelle als Einzige zulässig ist (hatten wir über *static $allowed_children* ... in der Datei *Categories.php* definiert). Zum Erstellen der Seite brauchen Sie nun nur noch auf *Go* klicken.

Nach diesem Schema sollten insgesamt drei Seiten angelegt werden. Wenn Sie anschließend eine der Kategorieseiten auswählen, können Sie neben den Tabs *Main* und *Meta-data* nun auch unseren neuen Tab *Products* mit der eben definierten Liste entdecken.

Sie können nun über den *Add Product*-Button neue Produkte anlegen. An dieser Stelle kommt das definierte Pop-up ins Spiel: Beim Hinzufügen, aber auch Ansehen und Editieren von Produkten öffnet sich ein Pop-up-Fenster mit einem passenden Interface.

Die hinterlegten Produkte erscheinen dabei in jeder Kategorie. Eine Produktzuweisung zur Kategorie der Seite erfolgt durch Markierung der Checkbox in der ersten Spalte. Markieren Sie also die gewünschten Produkte und klicken anschließend auf *Save & Publish*. Bei der Zuweisung werden Sie bemerken, dass zwar alle Produkte unter allen Kategorien sichtbar sind, sich jedoch nur die auswählen lassen, die noch keiner Kategorie zugeordnet wurden. Möchten Sie dies ändern, ist dies durch Korrektur bei der Definition der Abhängigkeiten sowie Auswahl der passenden Liste für die Ausgabe (beispielsweise *ManyManyComplexTableField*) ganz einfach möglich. Danach sind die technischen Arbeiten zur Verwaltung der Produktdaten abgeschlossen. Jetzt müssen die Daten nur noch ausgegeben werden.

MVC im Framework: Eine Performancefrage?

Für eine bessere Wartbarkeit der Inhalte und größere Flexibilität trennte man schon in frühen Zeiten der Applikationsentwicklung die Ausgabe von der Programmlogik, und es entstanden die ersten Template-, beziehungsweise CMS-Systeme. Aus Sicht der Performance war dies zwar ein erster Rückschritt, der aber aufgrund der neuen Möglichkeiten gerne hingenommen werden konnte.

Insbesondere aber bei der Programmierung sehr großer Anwendungen schien diese Aufteilung noch nicht ausreichend. Eine weitere Separierung mündete für Webapplikationen schließlich im bereits 1978 entwickelten und in Smalltalk implementierten MVC Design Pattern, welches ein Framework zur Verwaltung der aufgeteilten Bereiche (oft auch Schichten genannt) nutzt.

Das MVC Design Pattern beschreibt die logische Aufteilung der Webapplikation in 3 Schichten: Model (M), View (V) und Controller (C).

- Die Model-Schicht kümmert sich dabei um die komplette Datenverwaltung einer Anwendung. Über sie werden Datenbankrelationen definiert, aber auch Datenbankabfragen durchgeführt, deren Ergebnisse dann an die Controller-Schicht weitergeleitet werden können.
- Die Controller-Schicht kann nicht nur Daten von der Model-Schicht entgegennehmen, beziehungsweise anfordern, sondern nimmt auch Benutzereingaben von der View-Schicht entgegen und kann passende Inhalte für diese als Reaktion aufbereiten. Damit fungiert der Controller als Mittler zwischen View- und Model-Schicht.
- Die View-Schicht ist für die Ausgabe, beziehungsweise Darstellung im Browser verantwortlich und lässt sich am ehesten mit einem Template-System vergleichen.

Der Datenaustausch innerhalb kooperierender Schichten ist eindeutig und über spezielle Konventionen geregelt.

Oft genannter Kritikpunkt eines MVC-Frameworks ist die schwächere Perfomance, die bei starken Serverlasten schneller nach einer Skalierung verlangt als traditionelle Content-Management-Systeme. Diese Tatsache entfacht bei den Entscheidern oft eine Diskussion über die Wirtschaftlichkeit eines solchen Systems. Mit zunehmender Erkenntnis, dass die Personalkosten für Entwicklung und Betreuung einer Applikation den Aufwand für weitere Hardware bei weitem übersteigen können, kommt man nun aber doch auf die Vorteile dieser Technik zurück.

Ein weiterer, nicht zu unterschätzender Punkt ist die schnelle Realisierbarkeit, die gute Wiederverwendbarkeit und nicht zuletzt auch die Sicherheit einer Anwendung, denn bereits funktionstüchtige, vom Framework bereitgestellte Bereiche müssen nicht nochmals neu erfunden und getestet werden, sondern sind nach ein paar Anpassungsarbeiten kurzerhand einsetzbar. Gerade dies kann ein entscheidender Wettbewerbsvorteil sein.

Kritiker empfinden außerdem das Schichtenkonzept als zu starr und unflexibel. In der Praxis wird man beim MVC Design Pattern allerdings nur an Grenzen stoßen, wenn es um die Erfüllung wirklich ganz spezieller, exotischer Anforderungen geht. Die Tatsache, dass MVC auf Basis spezieller Vorgaben (*Convention over Configuration*) für minimalen und übersichtlichen Code sorgt, sollte in den meisten Fällen eher hilfreich und produktiv, und nicht als einschränkend empfunden werden.

Aktuelle Projekte bestätigen den Vormarsch des MVC Design Patterns als zukunftsorientierte Architektur. So beschäftigen sich nach Joomla! auch die TYPO3-Entwickler mit einer entsprechenden Implementierung. Allerdings sollte man sich schon vor der Planungsphase Gedanken in Bezug auf Funktionsumfang und Performance machen.

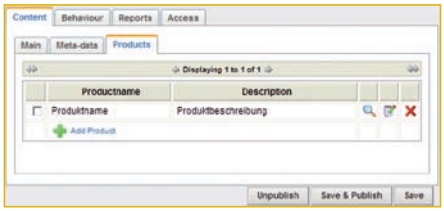

Abb. 5: Backend zur Verwaltung der Produktdaten

Wenn Sie jetzt einen Blick auf die Frontend-Ausgabe von SilverStripe riskieren, werden Sie feststellen, dass die Seite „Produktkatalog“ zumindest schon mal ein Untermenü mit den Kategorien ausgibt. Für die Kategorien selbst müssen wir die Ausgabe im Folgenden aber noch definieren. Ein Blick in die Datei */mysite/_config.php* verrät uns mit den letzten Zeilen, dass SilverStripe die Templates im Ordner *blackcandy/* nutzen soll. Die folgenden Änderungen beziehen sich also nun auf die Dateien unter */themes/blackcandy/templates*.

Im Ordner *templates/* befinden sich die Template-Dateien für einen Seitentypen. Die Zuordnung wird dabei automatisch über den Namen der Datei ermittelt.

Die Datei *Page.ss* ist damit automatisch Template für den Seitentyp *Page*. Sie enthält allerdings nur das Seitengerüst und beinhaltet die Elemente, die von jeder von *Page* erbenden Seite genutzt werden. Wenn Sie diese Datei öffnen, werden Sie neben gewohntem HTML-Code auch diverse Platzhalter erkennen. Die Platzhalter erscheinen im Template wie PHP-übliche Variablen.

Für den Platzhalter *$layout* wird der Inhalt des gleichnamigen Templates (*Page.ss*) aus dem Ordner *Layout/* eingebunden. Die Datei */Layout/Page.ss* enthält damit also den eigentlichen Inhalt des Seitentyps *Page*.

Immer wieder genutzte Elemente, die nicht automatisch auf Basis ihres Dateinamens eingebunden werden sollen, können übrigens im Ordner *Includes/* abgelegt und über einen *Include*-Befehl in anderen Templates eingebunden werden.

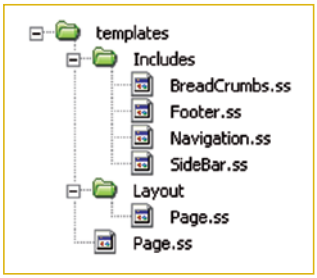

Abb. 6: Aufbau des Template-Ordners

Die Seite „Produktkatalog“ vom Typ *Categories* zeigt – anders als die Kategorieseiten – eine Seite im gewohnten Layout an. Das kann sie deshalb, weil der Controller die Funktionalitäten von *Page* geerbt hat (siehe */code/Categories.php*). Würden wir jetzt im Ordner *Layout/* eine weitere Datei mit dem Namen *Categories.ss* anlegen, könnten wir die Ausgabe für diesen Seitentyp neu definieren. Genau dieses Wissen wenden wir nun aber auf

Listing 1 Anlegen eines neuen Tabs

```
<?php

class Category extends Page {

 static $has_many = array('MyProduct' =>
                                    'Product');

 function getCMSFields() {
  $fields = parent::getCMSFields();

  $tablefield = new HasManyComplexTableField(
 $this,
 'MyProduct',
 'Product',
 array(
  'Productname' => 'Productname',
  'Description' => 'Description'
 ),
 'getCMSFields_forPopup'
);

  $fields->addFieldToTab("Root.Content.
                        Products", $tablefield );

  return $fields;
 }
}

?>
```

Listing 2 Product.php

```
<?php

class Product extends DataObject {

  static $db = array(
    'Productname' => 'Varchar(50)',
    'Description' => 'Text',
  );

  static $has_one = array(
    'Image' => 'Image',
               'MyCategory' => 'Category'
  );

 function getCMSFields_forPopup() {
  $fields = new FieldSet();
    $fields->push( new TextField('Productname'));
    $fields->push( new TextField('Description'));
    $fields->push( new ImageField('Image'));
               return $fields;
 }
}

?>
```

den Seitentyp *Category* an, denn für diesen können wir uns nicht auf bestehende Templates stützen, sondern müssen noch Änderungen zur Ausgabe der Produktdaten einarbeiten. Hierzu kopieren wir die Datei */themes/blackcandy/templates/Layout/Page.ss* und speichern sie unter dem Namen *Category.ss* direkt daneben ab. Nach dem Öffnen können Sie die Platzhalter *$Content*, *$Form* und *$PageComments* erkennen, die sich auf zugeordnete Funktionen von Model und Controller beziehen. Da wir die Ausgabe eines Formulars an dieser Stelle wohl nicht mehr benötigen werden, verzichten wir auf den Platzhalter *$Form* und fügen an dessen Stelle den Code aus Listing 3 ein.

In diesem Template-Code kommt unser bereits definiertes *MyProducts* als Referenz zu den Abfrageergebnissen zum Einsatz: Zunächst wird überprüft, ob Ergebnisse vorliegen. Wenn ja, werden diese über eine Schleife (innerhalb von *<% control ...>* und *<% end_control %>*) ausgegeben. Wenn nicht, wird eine entsprechende Fehlermeldung zurückgeliefert. Wie eben erwähnt, muss für eine korrekte Ausgabe jetzt nur noch unter */mysite/code/Category.php* geerbt werden:

```
class Category_Controller extends Page_Controller {}
```

Fertig. Nach einem finalen Refresh sollte nun auch endlich das Frontend des Produktkatalogs zu bewundern sein.

Fazit

SilverStripe macht Spaß: Alle, die bisher etwas neidisch auf Ruby on Rails geblickt haben, erhalten mit SilverStripe eine PHP-Alternative – inklusive CMS. Im Handumdrehen lassen sich mit wenigen Zeilen beliebige Datenbankstrukturen und passende Backends zur Verwaltung aufbauen. Sehr schön.

Auch die Einarbeitungszeit für Programmierer dürfte deutlich unter dem Niveau liegen wie man es von anderen Systemen, beispielsweise TYPO3, her kennt. Aber auch Redakteure sollten mit dem aufgeräumten Backend wesentlich schneller zurechtkommen und von anfänglichen Überforderungen verschont bleiben. Zwar merkt man dem System an einigen Stellen noch sein recht junges Entwicklungsstadium an, doch die freundliche Community und die engagierten Entwickler sind stets bemüht, zur schnellen Lösung eines Problems beizutragen. Wer ein einfaches Zusammenklicken fertiger Erweiterungen erwartet, erhält zwar zum jetzigen Zeitpunkt nur eine übersichtliche Auswahl an Modulen, allerdings richtet sich das System mit seinem Konzept auch eher an Programmierer, die auch mit weniger professionellen PHP-Kenntnissen eine schnelle Anpassung an individuelle Wünsche vornehmen wollen.

Unterm Strich hält SilverStripe sein Versprechen: Einfachere Installation, einfachere Einarbeitung und einfachere Erweiterung. Der Exot aus Übersee schafft den Spagat zwischen Einfachheit und Machbarkeit und garantiert damit in wesentlich kürzer Zeit eine individuellere Anpassung an die Kundenbedürfnisse als viele andere Systeme. SilverStripe ist also wirklich ein farbenfroher Schmetterling, der gerade erst begonnen hat, mit den Flügeln zu schlagen. Wir sollten seinen Flug im Auge behalten.

Listing 3 Template-Code in Category.ss

```
<% if MyProduct %>
 <% control MyProduct %>
  <div style="border-bottom: 1px solid #000;">
        <p>Productname:
<strong>$Productname</strong></p>
       <p>Description: <strong>$Description</
strong></p>
       <p>$Image</p>
  </div>
 <% end_control %>
<% else %>
 <p>Kein Produkt vorhanden!</p>
<% end_if %>
```

Steven Broschart (pm@broschart.net) ist bereits seit mehreren Jahren als Core-Programmierer und Senior SEO Consultant für den Online-Marketingspezialisten cyberpromote tätig.

Links & Literatur

[1] Butterflycorner: www.butterflycorner.net

[2] silverstripe.com