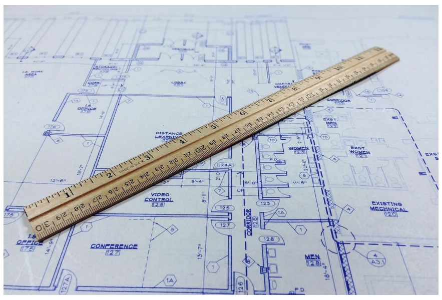
ALLAN THRAEN | 6 years ago | PDF |

CMS Contentful

THE CURIOUS CASE OF CONTENT MODELLING



Having the right content model (the structure of your content types) is very important in order to end up with good, usable (and reusable) content. I believe that is something that most content management aficionados can agree on. But what is a good content model? And who should be modelling your content? In this blog post I will try to discuss a few opinions on this topic.

Whenever I visit a new client for the first time and try to get an overview of how they use their content management system, I always start by asking to see their content models as it's usually a pretty good indicator of the overall health of their content.
I might be hard to say, exactly what *the right* content model is for any given case, but it is fairly easy to spot when it has gone wrong.
For example, when I see 50+ page types in an Episerver installation, with a naming pattern similar to "Article Page - 3 columns", "Article Page - 2 columns" and so on, then I know that the experience delivery has completely taken over any human sense in the content modelling.
It will be quite a challenge for a content worker to decide which page type to use - and if they choose wrong, changing the type without content loss isn't always easy. And any future attempts to streamline or convert the site will be very expensive as content will be very diversified across many different models.

Equally scary is it, when you stumble across a large international enterprise with only 1 content type that has only 1 property, asides from the name, a full page XHTML field.

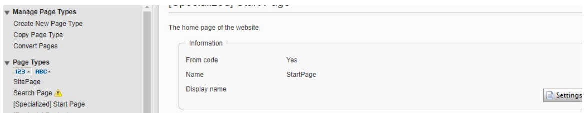
```
[ContentType(DisplayName = "SitePage", GUID = "da5cc2e1-ad8b-42c8-a6a0-328fa109fa49", Description
public class SitePage : PageData
{
    public virtual XhtmlString Body { get; set; }
}
```

I can of course see why developers find it appealing at first; less code to write = fewer bugs = less code to maintain. At the same time you could argue it gives great freedom to HTML-savvy editors. But it's a disaster in the making.
I remember visiting a customer with such a setup. The editors said they didn't mind. They had made 'a system' to make content production easy; a 'secret' part of their page tree contained a lot of pages that worked as templates. For example, if they needed a 3 column design for a page, they would simply find the 'template' page called "3 text columns" which contained a good old HTML table with 3 columns in it's XHTML field, they would copy it and simply modify the contents of the columns in the table...Some times I still wake up at night, screaming and covered in sweat after having a nightmare about this approach. But enough about me :-)

Developers vs Editors - Who's best at Content Management?

The situation described above does raise an interesting question: Who should be modelling your content? The editors/content producers? A librarian? An external content consultant? The developers?

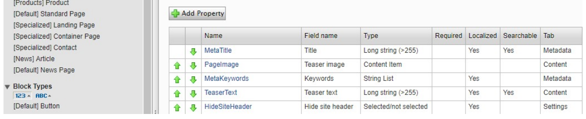
In Episerver CMS, content type definitions was originally something that was stored in the database and managed through Admin mode - typically by the developers or super users as part of the site construction.



The home page of the website

Information

From code	Yes
Name	StartPage
Display name	


[Products] Product
[Default] Standard Page
[Specialized] Landing Page
[Specialized] Container Page
[Specialized] Contact
[News] Article
[Default] News Page

Block Types
[Default] Button

Add Property

		Name	Field name	Type	Required	Localized	Searchable	Tab
	⇩	MetaTitle	Title	Long string (>255)		Yes	Yes	Metadata
⇧	⇩	PageImage	Teaser image	Content Item				Content
⇧	⇩	MetaKeywords	Keywords	String List		Yes		Metadata
⇧	⇩	TeaserText	Teaser text	Long string (>255)		Yes	Yes	Content
⇧	⇩	HideSiteHeader	Hide site header	Selected/not selected		Yes		Settings

While you can still manage content types through Admin mode, it really all changed when Joel Abrahamsson introduced PageTypeBuilder (PTB) in 2009.
PTB was a plugin that allowed developers to define their content models as types in their code, describe characteristics of each property using attributes, use inheritance of content models, work with strongly typed models throughout the rest of their code (like calling 'CurrentPage.MainBody' as opposed to 'CurrentPage["MainBody"]') and use source control to manage content models alongside the rest of the site code that depends on it. Many great advantages!

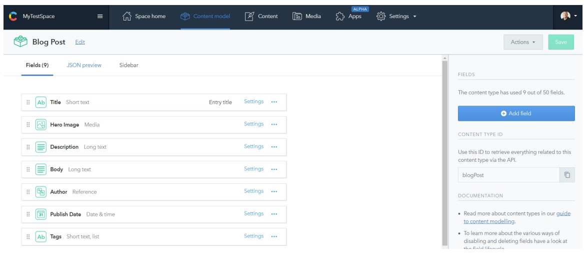
It quickly gained a lot of traction and by the time Episerver 7 was getting released, 90% of new installations used PTB, so Episerver built-in similar functionality in the core product - and that has been the prefered approach to modelling your content since in Episerver.

But this also means that often the responsibility of making decisions around how the content should be modelled will fall to developers that often are experts at building web-sites, but not necesarilly experts in the business domains - and hence will lean towards a very web/delivery centric content structure...

Until recently I was fairly certain that content modelling in code was the best approach - in spite of the little voice in my head reminding me that it's not always a good idea to let *the inmates be running the asylum*. However the benefits clearly outweighed the negative aspects, and could of course be mostly remedied by having developers like myself and others that are passionate about web content management and business use cases carefully model the content in collaboration with the site's business users. But a recent experience has made me question that logic a little:

I recently attended a Contentful meetup where a large contentful customer was presenting a case story of how they work with their own web site and app. The customer had sent both a developer and a business user to talk about their experience - and what almost made me fall down the chair was that it was the business user that owned the content modelling!
And to make matters even more strange, they both seemed to enjoy that division of work. The business user was part of the communications team that produced most of the content and managed the web site. And as they knew exactly what business needs their content should adress, they simply modelled it to fit their needs.

If they for instance added a new field, or even a new content type, it would of course not be visible on the web site until their developer had put it into their rendering, but they could start to create content for it right away and that way it would be easy for the developer to see just how it would fit and make the proper front-end adjustments. In fact, all they needed was a front-end developer to update their site with the desired changes, as there wasn't really any backend connected to the content.



Blog Post Edit

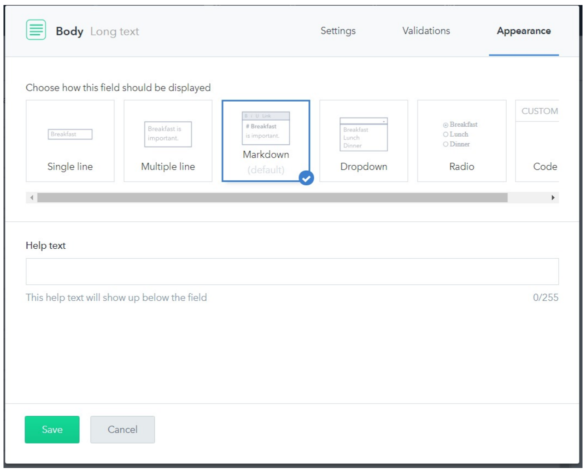
Title	Short text	Entry title	Settings
Hero Image	Media		Settings
Description	Long text		Settings
Body	Long text		Settings
Author	Reference		Settings
Publish Date	Date & time		Settings
Tags	Short text, list		Settings

FIELDS
The content type has used 9 out of 50 fields.

CONTENT TYPE ID
Use this ID to retrieve everything related to this content type via the API.
blogPost

DOCUMENTATION
- Read more about content types in our guide to content modelling.
- To learn more about the various ways of disabling and deleting fields have a look at the field lifecycle.

This approach of course entails that you don't delete fields/properties already in use by a developer on a site - which is why it's fairly hard to remove fields from a content type once it's added and in use. And you have to go through several warning levels. Just as super-user content workers can create content types and fields they can of course also configure the fields and decide how it should be presented (think Episerver UIHint).



Choose how this field should be displayed

Single line | Multiple line | Markdown (default) | Dropdown | Radio | Code

Help text

This help text will show up below the field 0/255

Now, Contentful actually does have something like PageTypeBuilder - but I don't know how much it is used. But they also have the reverse - a way to create code classes from your defined content models so you can still enjoy the benefits of strongly types content models.

I still don't know if it is better in general to have content workers define the content models or let the developers do it in code. And in some degree it of course depends on exactly which resources you have doing the jobs - and what skill-level they have. But as a content-nerd, I do find it quite an interesting discussion to have.

What is a good content model?

A closely related discussion is of course also what a good content model actually is. I generally prefer to keep content models focused around the business types they represent:

- Article
- News Item
- Product
- Blogpost
- Event
- Teaser / Feature

and so on. And ideally I try to seperate the models that keep actual content separated from models that describe channel/delivery specific information. Cause even though we might want to avoid it, it can be very hard to keep out of your content management system.
Some examples of channel/delivery specific content could be:

- Landing and listing pages
- View formatting properties (ShownOnLeft / ShownOnRight)
- Side bar navigation
- Editor controlled HTML/Script injections
- Analytics tracking ID's
- Different column views
- Forms
- Top logo image
- Navigation menus

Ideally some of these settings could be induced directly in the site development, or implied by additional view data - like the "Display As" option in an Episerver Content Area property. But you can never avoid having some channel specific properties exposed to the editors. In those cases I simply suggest you separate it so it doesn't pollute your clean, resusable content.

In Episerver, one approach for this could for instance be to keep clean re-usable content as blocks, but have pages managing the web delivery. It's not the only solution - and not always the right solution, but worthy of consideration - it might be right for you.

How do you feel about content modelling? Share your thoughts in the comments below!
And - if you'd like to have me visit and discuss your content model or take a look at your site, my contact info is here.

Update 2019-04-08: Deane Barker wrote a nice rebuttal to this blog post with his views on the matter.

CMS Contentful

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688