

# The making of ERS 2.0

---

*Getting Started with Cloud CMS*

2017



# The Making of ERS 2.0

---

*Getting Started with Cloud CMS*

2017.

**CONTACT**

Samuel Pouyt

[samuelpouyt@gmail.com](mailto:samuelpouyt@gmail.com)

# Executive Summary

The European Respiratory Society (ERS), a non-profit organisation, has, lately changed its approach on managing its content to accommodate a rather disparate technological landscape: CMS's in PHP (Joomla, Drupal), another in .NET, a custom CRM written in .NET, many small apps, SAP, some servers on premises, others in the cloud, legacy systems, etc. After more than 20 years of existence, the ERS has more than 25 websites that relate to its activity and other projects, a small communication and marketing team, as well as a small IT team.

For a small team, it is difficult to manage all those websites as a lot of time is just spent in maintenance, upgrades, security patches and other migrations. There is not much time left to improve the existing websites and develop new features. That is without saying that there are entirely new website to create. The user experience suffers from this diversity and as time passes the quality of the digital experience deteriorates: broken links, missing pages, outdated content, and the worst legacy technologies that prevent the whole stack from any further improvements.

Due to the nature of common CMSs the content was tightly coupled with the technology necessary to display it. Thus any change of website implied complicated content migration and a lot of copy pasting. Mobile apps, need to access the content created in the CMS thus a way for the app to connect to the CMS to retrieve the content needs to be implemented. This adds another layer of maintenance and an additional source of bugs.

Therefore, we decided to look towards the future and change our approach. The content, whether it is news, products or anything else is center to any website, mobile app, and any other platforms (Facebook, Twitter, etc.). Thus we chose a product that only manages content, and that is doing only this, but that does it very well. It does not influence the design nor the technology we want to use to display that content. It is IOS, Android. PHP, Node, .Net, etc. friendly. It can be fully integrated in automated workflows. It is a Headless CMS: Cloud CMS. This type of CMS does not do any assumption on technology or design, it does not display content, it just serves content as JSON.

We currently have a central point where we manage our content, that we do not have to maintain and that provides a customisable administration interface for our content editors. We are now able to seamlessly integrate our content within different technologies. We could also reduce the time we spend on maintenance to improve the quality of our user experience. We have reduced the amount of website we need to manage, but we are not yet done. Indeed, we need to migrate other websites, but we also plan to add some automation as, for example, AI services to extract keywords, suggest hashtags, extract topics, a recommendation engine, and many other features.



It was long over due for the European Respiratory Society — one of the leading medical organisations that brings together physicians, healthcare professionals, scientists and other experts working in respiratory medicine, with a growing membership representing over 140 countries worldwide — to revamp its main website as well as to start standardizing its brand in the digital sphere.

Two solutions were offered to us: do as usual or look towards the future. We are a small team managing around 25 websites. Some are purely informational, some are event based. Every year we have few websites that are decommissioned and new ones that need to be created. We also have a member portal — MyERS — where members can create an account and manage their activities within the ERS, a CRM — the admin side of the member portal, an abstract platform, an event management platform, SAP to manage our accounting, etc.

All these application need to communicate among themselves to variable degrees, whether it is sending payment information to the accounting system, check if the user is a member or simply display content in more than one place e.g. news on the main website or in an app. Changing the main corporate website allowed us to solve the content management issue to accommodate, the IT team, but also the marketing and communication teams. Indeed changing our content management strategy helped us save a lot of time and simplify our content editor's lives.

**The usual  
way**

Until now we have always developed websites using a CMS, mostly Joomla! to create new websites. We know that CMS inside out, and we are really able to produce websites quickly as we have a set of extensions that we have vetted and that solve most of our typical use cases.

Going that direction does not require much thought. We just have to concentrate in the design, migrate the content and we are all set. Our users know the CMS and they can, with no training, start using the new website right away. For few years we have worked this way, and we know it works, but it is not an optimal long-term strategy.

## Starting a project

The CMS needs to be installed, then all the extensions required by the project. Usually, as the project is new, the latest versions of those extensions are installed. We never managed to create a starter project, as it was not up to date almost as soon as it was created. And that was an additional element that we would have needed to maintain. After a few years, all our websites were using different versions of those extensions as you never find time to go through all website to update those. Indeed, if you want to do these updates in a safe way, ideally you would have a test version of that website which you would update before publishing the changes to production. Git and modern CI (continuous integration) workflow do not really help either as many settings (in Joomla! at least are saved to the database) thus you cannot always just pull your new files in the

production environment as you need to “install” that update or patch, or run some database migrations

## Security patches

The other problem is security updates. Those open source CMSs (Joomla!, Wordpress, Drupal, etc.) are very well-know by hackers too, thus security patches need to be constantly implemented, that is if the website was not already hacked. On one website that is ok. But when multiplied by 25, the process of patching eat up a lot of development time. It is time consuming because to patch a website, you have to make sure that the patch is compatible with the installation on which it is deployed. To make sure that everything works as expected, the patch needs to be deployed first on a testing environment, then deployed in production.

## Updated or deprecated extensions

Additionally, extensions have their own update cycles. Sometimes it might happen that they are not supported anymore or they are not upgraded to be compatible with the latest version of the CMS... We tried to mitigate this risk by choosing extensions backed by reputable company. But, nevertheless, those companies have their own plans... It has happened that we had a website that relied heavily on such extensions and when the extension was not

supported anymore, it prevented us from updating the CMS making us face three possible choices: migrate our content to a different extension, write our own extension based on the one that was not supported, or do nothing and have an aging website.

## Custom code

The code we write is not really reusable even if we create installable extensions (plugin, components or templates). It is difficult to share the code with other websites as, very often the underlying database schema is different; thus, to manage all the edge cases it is sometimes inevitable that the extensions become extremely complex.

And if we create a template package that we use on many websites, we then need to update it on each website when we want to change something... Very quickly, we end up having discrepancies on all the different websites. Indeed some custom component need to work slightly differently on different website and soon enough you have two or more versions that you need to maintain.

## Users

We also need to maintain access rights to all our CMSs. The solution we chose was to keep it simple and to create accounts upon request. This choice of having multiple CMSs made sense at some point as we could have a website up for few month and take it down without impacting anything else. On the other hands, our editors need to login in multiple administration interface, and remember the small nuances that each one of them is providing. We tested some multi-site extension, but they just complicated maintenance as Joomla! Is not really build for it. There are of course better CMSs than Joomla! to manage this use case but by that point in time our choice had already been made.

## Design

The main idea with the redesign of the ERS website was that we wanted to separate concerns: the design would be distributed from a central point and would have nothing to do with the website implementation, thus we would serve our CSS and JS so that any website could use it and we could update all websites at once.

So after multiple iterations on sketch and photoshop and concept approval by stakeholders we created a custom version of bootstrap implement-

ing our designs. We started distributing it. Thus all new websites started using it and we could manage it with a simple workflow to publish all the assets. Our Bootstrap has its own Github repository, can be tested separately and new versions can be published on their own. It is now easy for us to add new design features to all our websites. Doing so we solved one of our maintenance problem. We do not need anymore to create installable templates that we need to install on each website. We just plug our CSS and Javascript.

However, it was not only challenging to manage the design, but also the content... For example, we have courses, that we advertise on the main website, but the registration takes place on MyERS. We display news on the main website, but also on other websites and apps. Much of the content is — at least in part — duplicated. For example our content editors need to create courses in two places: one that manages prices and checks the user status (MyCRM), the other — where all marketing attributes and other descriptions are added (CMS).

If we were able to distribute the design across websites from a central point, wasn't it possible to distribute the content from a central point as well?



# **The Headless Meet Cloud CMS**

# What is Cloud CMS

POST	/repositories/{repositoryId}/branches/{branchId}/nodes/{nodeId}/change_type
PUT	/repositories/{repositoryId}/branches/{branchId}/nodes/{nodeId}/change_type
POST	/repositories/{repositoryId}/b/{branchId}/nodes/{nodeId}/change_type
PUT	/repositories/{repositoryId}/b/{branchId}/nodes/{nodeId}/change_type
POST	/repositories/{repositoryId}/branches/{branchId}/n/{nodeId}/change_type
PUT	/repositories/{repositoryId}/branches/{branchId}/n/{nodeId}/change_type
POST	/repositories/{repositoryId}/b/{branchId}/n/{nodeId}/change_type
PUT	/repositories/{repositoryId}/b/{branchId}/n/{nodeId}/change_type
GET	/repositories/{repositoryId}/branches/{branchId}/nodes/{nodeId}/lock
POST	/repositories/{repositoryId}/branches/{branchId}/nodes/{nodeId}/lock
POST	/repositories/{repositoryId}/branches/{branchId}/lists/{listKey}/items/query
POST	/repositories/{repositoryId}/b/{branchId}/lists/{listKey}/items/query
DELETE	/repositories/{repositoryId}/branches/{branchId}/lists/{listKey}
GET	/repositories/{repositoryId}/branches/{branchId}/lists/{listKey}
POST	/repositories/{repositoryId}/branches/{branchId}/lists/{listKey}

» Fig. 1 - Few API endpoints

Cloud CMS website we can find the following description. “Cloud CMS is a headless, API-first approach to content management, built around JSON and a high performance cloud architecture. It delivers enterprise features, including flexible content models and a full editorial environment, allowing your business users to create, manage and publish fresh content with ease.”

## Content is just... content

This is what a headless CMS, or an API first CMS does. It manages content, and this is the only thing it does... but it does it well. It does not have any impact on design, it just serves content via an API. The only thing that the CMS returns is JSON (or binaries e.g. documents, images). Actually everything in Cloud CMS is JSON: configuration, models, forms, relations. Cloud CMS API lets you manage any aspect of the platform and retrieve everything as JSON.

## Multi-device, multi-technology

JSON is like the English language for scientific papers, but for applications. It is the lingua franca of the internet. As far as everything is JSON in Cloud CMS the content is de facto multi-platform, multi-technology, multi-device. Cloud CMS lets you get content with its powerful API but it also lets you write to it.

At ERS, we can from our custom CRM written with .NET create a course: our business user input all the data that the CRM needs, the CRM can push this data to Cloud CMS and programmatically create an article. The content editor can then add marketing information using the

Cloud CMS administration interface to the newly created article. Then full content, price and marketing content can be displayed anywhere, on the main website, in an app, in the members portal etc. This really improves the workflows for all teams at ERS, as different people can seamlessly work together, saving time but also insure a better quality content. The content is really centralised, versioned and easily accessible. Here is how our infrastructure looks like at the time of writing:

```
Document JSON
Home / documents / ERS_Excellence_Award_for_Research_in_Cystic_Fibrosis
ERS Excellence Award for Research in Cystic Fibrosis
Article (ersarticle)
Code
1- {
2  "title": "ERS Excellence Award for Research in Cystic Fibrosis",
3  "slug": "ers-excellence-award-for-research-in-cystic-fibrosis",
4  "contentType": "event_award",
5  "flags": [
6    {
7      "text": "Applications closed",
8      "color": "warning"
9    }
10 ]
11 "leadParagraph": "This research award of 7,500€, financially supported by Vertex, is awarded in recognition of research and Medicine. The award is exclusively reserved to scientists and clinicians. The award prize is to be used to . . . The award winner will be awarded at the 2017 ERS International Congress in Milan (9-13 September 2017) a related scientific session.",
12 "body": "### Criteria\n\nEligible candidates should be ERS Members and should never have won an ERS Award prize last three years. Candidatures of young researchers are encouraged, however the career-stage will prevail. The award will be made by the ERS Science Council. The Award's applications are judged independently of the sponsor on the How to apply\n\nApplications/nominations for 2017 are now closed.\n\nRead the [Guidelines for the CF R /o0p5vignlp3y0yhk93rf1l6z7ljnsc)\n",
13 "articleOneColumn": true,
14 "itemImageAlignment": "center",
15 "itemImageBackgroundSize": "100%",
16 "imageSize": "small",
17 "category": {
18   "ref": "node://18dbd4f08d5f428ba9c2/607e97e4474d46e40345/b6b2871b2d9cf6b4996b/76ee36f7b7b17eebace9",
19   "id": "76ee36f7b7b17eebace9",
20   "title": "Research Excellence",
21   "qname": "o:76ee36f7b7b17eebace9",
22   "typeName": "ers:category"
23 },
24 "category2": [
25   {
26     "id": "93cd508f81fa78c83cd2",
```

» Fig. 2 - Partial JSON document

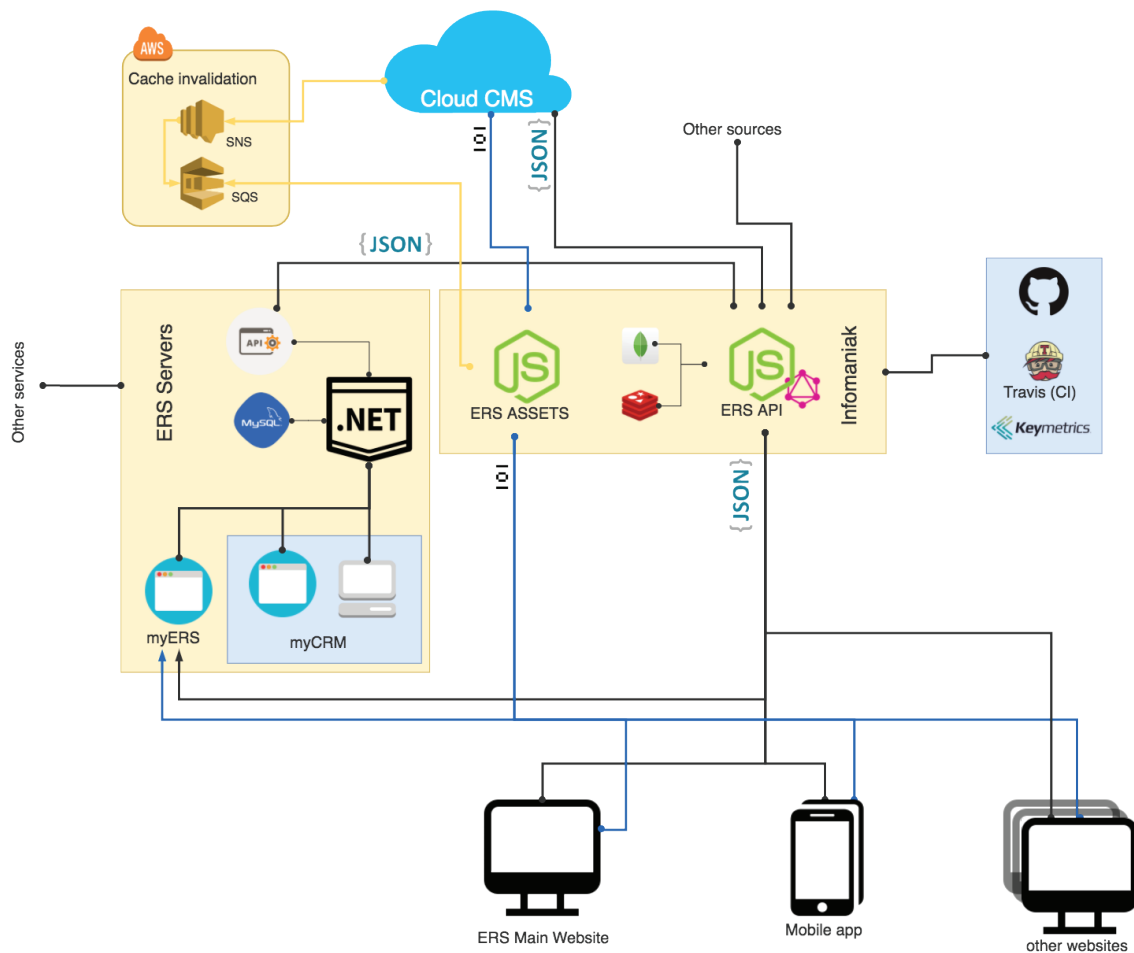


Fig. 3 - Cloud CMS integration at ERS

## Development

At first sight it seems that choosing a headless CMS will greatly increase time needed to develop the website as the whole user interface/experience (UX) needs to be developed from scratch. Indeed, before, we were always starting a website from a “base”: the fresh installation of the CMS, that we were tweaking for our needs.

But in fact, it was a very liberating experience to start from a white sheet (or a black screen). Our code base is much cleaner, as we have only the functionality we need, we do not have code that we do not use, as it is part of the functionality that is shipped with the CMS. This approach gives a lot of freedom to the development team, whether it is an agency or an in house team. In fact, when you need to develop a new website, you can hire subcontractor

and provide them the API documentation, API keys and they can start using your content right away! There are no more content migration, synchronisation or replication.

The web development world evolves very quickly, new standards appear, new frameworks. This is not a problem, if the developers want to transform the front-end and use the last framework à la mode, they have that freedom. The developers can even have part of the website using one technology, while the other uses another. Developers loves this freedom as they can always choose the technology that works best for a given task. That would have been difficult to achieve with a traditional CMS.

## Time saver

It might seem that a headless CMS is more difficult to put in place and that you will need more time and resources, because you need to develop everything from scratch (Cloud CMS provides seed projects for different technologies to get developers started). This is true to some extent, if we compare the time needed to deploy a CMS with a ready made template that was slightly adapted to our needs. In the past we have deployed brand new websites in 3 days. But these website are not complicated and are not very customised. They also feel like the CMS that was used to create them in the first place.

As we have seen, “erasing” the look and feel of the CMS takes a lot of time as it is necessary to rewrite all the outputs (views) of all the components, plugins and module installed on the system. Otherwise the website does have a look and feel of Joomla!, Drupal, or Wordpress. This might be acceptable, unless you want your brand to be coherent.

boilerplates are ready, it takes no time to create new pages.

For the main corporate website of the ERS with the brand new design we planned 10 days for the integration into Joomla!, while we planned 6 for the headless CMS. 4 days do not sound like a lot, but it quickly increases as soon as we started adding new features. With our implementation of Bootstrap ready, we managed to have a fully working website in less than 25 days of work. As for every project, we presented it to stakeholders, reworked it a little and we were ready for launch.

A standard CMS also comes with many features you do not need as it needs to cover a broad range of use cases. This is also true with Cloud CMS, you have many features like content workflows, versioning, task management that help teams collaborate on content, but these features never get in the way nor for the user, nor for the developers. When the content is ready to publish, it is just content and it can be retrieved as plain content.

```
/**
 * Get a single article (or category) by slug
 * @param string $slug (slug-of-the-artice)
 */
public function getItem($slug){
    $query = '{"slug": "'.$slug.'"}';
    $result = CC::nodes()
        ->query($query)
        ->addParams(['full' => 'true'])
        ->addParams(['metadata' => 'true'])
        ->get();
    $result = $this->validateResults($result);
    return $result;
}
```

» Fig. 4 - PHP method to get an article based on its slug/alias (part of the url)

This is a lot of work and it takes a lot of time. Sometimes components do not implement very well with the MVC (Model View Controller) structure of the CMS, thus core files of the component need to be modified and the update is not anymore just a click on a button or a simple package install: a migration is need in order to preserve the design modifications.

On the other hand, starting with a white page can help saving a lot of time by developing personalized views and partials that can easily be reused. Yes, the initial workload is bigger but when the first

The screenshot displays the European Respiratory Society (ERS) content management system interface. At the top, the ERS logo and name are on the left, and navigation links for 'Help', 'Editor', and the user 'Samuel Pouyt' are on the right. Below the header, the breadcrumb path reads 'Platform / Projects / ersnet.org Main Website / Documents'. A search bar is present on the right side of the breadcrumb area.

The interface is divided into three main sections:

- Left Sidebar:** Contains navigation options under 'CONTENT' (Content, Documents, Tags) and 'DELETED ITEMS' (View Deleted Items).
- Document Properties Panel:** A vertical menu on the left of the main content area with options: Attachments, Edit Properties, Graph, Tags, and Versions.
- Main Content Area:** Titled 'Document Properties', it shows a 'Home' link and a document preview for 'Beta-blockers could reduce the risk of COPD exacerbations' (Article (ers:article)). Below the preview is a 'Save' button.

The 'Article' section contains the following form fields:

- Title:** Beta-blockers could reduce the risk of COPD exacerbations
- Sub Title:** (Empty)
- Slug / Alias:** beta-blockers-could-reduce-the-risk-of-copd-exacerbations
- Content Type:** Article / Blog (dropdown menu)
- Type:** News (dropdown menu)
- Flags:** (Empty) with an 'Add New Item' button.

Fig. 5 - Simplified interface for content editors

**Technology**

# What could be better (and it will)

Cloud CMS can be at first overwhelming. For us the learning curve was steep, and we are still learning. Conceptually Cloud CMS is different, as you will see below, the underlying structure is a content graph. It changes the way you think about content and the way it is modelled. Of course, it is possible to reproduce a traditional structure, but there are much more efficient ways to work with, create and query content from a graph, but this needs to be learned... It is also necessary to learn the Cloud CMS vocabulary: definitions, features, associations, relators, etc. There is documentation, but it is as huge as the functionalities of the CMS.

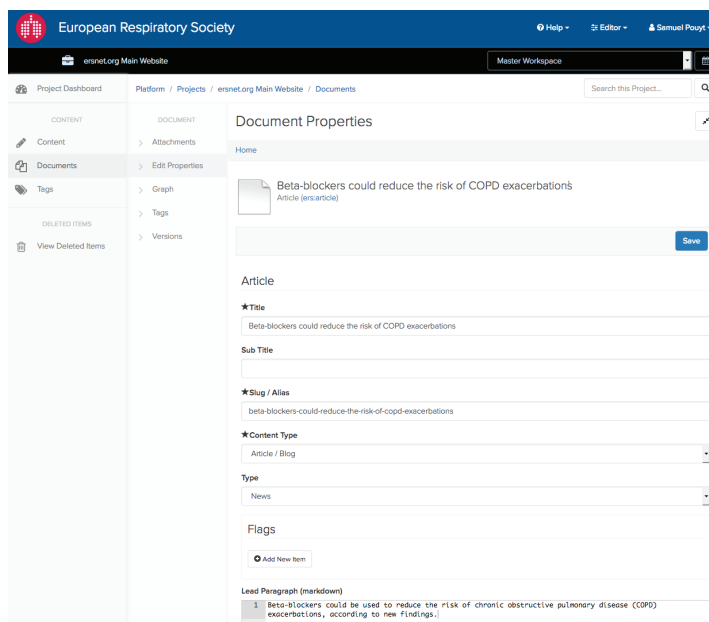
Every aspect of the editor interface can be customized. It is good, but at first, it is also overwhelming. The look and feel of the interface is very sleek, but, in some area, it has a technological touch, a geeky feel: the impression is that everything can be customized. This was a little bit scary at first, until we found our way.

The Cloud CMS team is continually working to improve the user experience and functionalities, the team has helped us a lot and implemented new features that we had suggested. Thus many aspects that we have mentioned are getting better and simpler.

All these options allowed us to provide the users with a custom experience as well as a simplified interface that corresponded to our needs but it also

allowed us implement all the enterprise features we could imagine: complex workflows, versioning, releases, granularity, etc. As a result, the we found that the user interface of the CMS is well thought through and, with experience we know that we can teach a new content editor in about an hour and the transition from Joomla! was very easy for them.

All the previously mentioned points are linked to Cloud CMS's own successes: an enormous set of functionalities that somehow they have to give access to...



The screenshot displays the 'Document Properties' form within the European Respiratory Society CMS. The interface includes a top navigation bar with the logo and user information, and a left sidebar with navigation options like 'Content', 'Documents', and 'Tags'. The main content area shows the article title 'Beta-blockers could reduce the risk of COPD exacerbations' and a 'Save' button. Below this, there are fields for 'Article', 'Title', 'Sub Title', 'Slug / Alias', 'Content Type', 'Type', and 'Flags'. A 'Lead Paragraph (markdown)' section is also visible at the bottom.

Fig. 6 - Form to edit an article

## MongoDB

One of the excellent features of Cloud CMS that is the fact that it uses MongoDB to store content. This schema less document store is very flexible as it only saves the the used properties and new properties can easily be added later on.

It also helps migration as the data can be of any shape. It is much simpler than editing the schema of a traditional relational database and you can do it right from the CMS, with a visual interface or right in the JSON, but this is not all.

Content can also be queried through the API using MongoDB operators such as “\$in” or Elasticsearch queries can be used which guarantee that the search queries are as quick and flexible as possible.

## Content Model

The content model is defined using a JSON schema. The schema can be created, updated, deleted with the API, but the administration interface let you quickly update it as well. The content model can inherit properties from other models and you can add new properties with “features”. Using features we add different options to all our content instances without having to maintain them inside the content model. For example we use a publication features that adds a check box to let the user unpublish an item or, for events, we add latitude and longitude in order to display a map on the website or to search by location.

## Forms

It is as easy to create or edit forms as it is to edit the model. The forms engine is called Alpaca and it is an open source library maintained by the Cloud CMS team. It can render html forms based on a JSON schema. It is quite easy to use and very flexible. We use a very complex form that displays different fields based on the content types, but the underlying content model is the same. Thus we are able to simplify the experience of our users and keep a consistent data model.

We have chosen to use Markdown as the format to input text. Our editors were used to it as they manage projects on Basecamp thus the transition was not complicated. We have taken this decision to prevent users from inadvertently breaking the layout of the website. But of course Cloud CMS has a classic WYSIWIG editor available.

## Graph

For us, starting working with Cloud CMS was a little bit surprising. Used to work with relational databases we had to face new technologies and concept that are not common among CMS's. We knew that we would be using a NoSQL database, but we discovered that actually we had to deal with a graph. The documentation that mentions OUTGOING and INCOMING relations pointed us into that direction. We had no idea what were those, thus after some googling, we were introduced to graphs and its theory. (This is not necessary to work with the CMS, but it is very interesting...)

In its implementation of MongoDB, Cloud CMS uses many concepts of Graph databases. Thus you can not only query documents based on some parameters, or index, but you can also query documents based on the relation, and the direction of this relation from on document to another or others.

Fig. 7 - Graph displaying an article (center) and all the incoming (←) and outgoing (→) relations

Although very useful, this approach challenges the usual understanding of items and categories as an article which has a “parent” relation to another article can be a node that links to many articles. The parent article becomes the de facto category, but it is just an article. Thus it is easy to query all articles “authored” by someone, or all the document related to another document. The relations are by themselves JSON object, thus they can be enhanced with their own properties.

## Versioning

All content in Cloud CMS is versioned. Thus the user can easily go back to a previous version or compare changes between versions. Cloud CMS even features the ability to branch content. It works like Git. But through a visual interface for the content editor, and of course you can access those features through the API. Branching is a powerful mechanism that let the user publish batches of content at a certain point in time. New content can be published when a branch is released (merged to the master), but a release can as well modify existing content and merge the branch to the master branch at a future date. This is really powerful as you could imagine a news website, that published a very short article for a breaking news, while an extended version of that article is being written, and approved, and then released.



