Roadmap v3

A lot of people have been asking for it for a long time. I think it's time now to reveal what will be the next development for 2014.

First, we learned a lot from our experience with the LocomotiveHosting platform. We got a lot of feedbacks from our clients and their own clients, especially concerning the UX/UI of the back-office.
Since we use ourselves LocomotiveCMS to craft sites for other companies, we see the big improvements we could add to our solution so that building and publishing a site become the easiest ever.

These are the 2 axis we are focus on

1/ WagonApp (Desktop version of Wagon)

The experience of installing Wagon is painful right now because it requires a lot of technical steps which may discourage people just discovering LocomotiveCMS.

Thus, new LocomotiveCMS users should install Wagon without launching the Terminal app or without installing the XCode tools. This app will also benefit to all the existing Wagon users.

Our goal is that a new user installs Wagon, create a site from a template (Blog, Portfolio, ...) and deploy it to the hosting platform in less than 5 minutes.

That's why we started to develop the Desktop version of Wagon, embedding Ruby and the Wagon gem. This also comes with a nice looking UI (see the screenshots).

How far are we from launching a beta version? Well, because we are learning a totally new technology (Cocoa), things take more time than expected so it's difficult give a fair estimatation. Anyway, in the development version we've got so far, you can list your Wagon sites, create/ remove sites, launch the Rack server and preview each site, see the output logs. The "deploy" functionnality should come soon.

2/ The Engine

We are going to move the CMS to a higher level.

a/ A single rendering component for both Wagon and the "Engine"

One of the major issue we've faced for the last months is that we need to maintain 2 projects, Wagon and the Engine. They share a huge part of the same code or at least same functionalities. So, basically, if we find a bug in the Engine, chances are big that this bug also exists in Wagon. Worst, sometimes, they behave differently for a same functionality.

So, we decided to extract the rendering component from the Engine and Wagon. That component will be based on a Rack application, a solution we chose for the current version of Wagon. It will also include the default liquid tags / filters and drops.
Each "rendering" functionality of the CMS (routing, locales, templatized page, ...etc) will be modelized as a middleware.

The good thing here is that modifying the way pages are rendered becomes easy. Just plug the new middleware in the stack, add the new liquid tags or filters and that's it. Good news, it will work, of course, for both Wagon and the "Engine".

The rendering component would be called "Engine" and the current Locomotivecms engine would be renamed to "Station".

We will also put a lof of efforts in the routing system, allowing developers to use fancy routes like in Rails. For instance, "/archives/:year/:month" will be handled by the new Engine.

b/ Station (the old engine)

Station is the back-office for the end-users, meaning it will just be the UI to manage the content of each site. It remains a Rails application but we will upgrade it to Rails 4.0 which should please a lot of LocomotiveCMS developers.

The only big functionnality we will remove here is the ability to edit a page template. The reason is simple, we don't want people to have different versions of a page template, one in Wagon, the other one in Station. That makes synchronization a nightmare. However, we won't go against developers if they build a gem to enable it. But we won't code this gem for sure :-)

We are going to implement what we call "the global site variables". Think about custom fields but applied to a site. Basically, we can add a "phone number" field to a site and this will be editable in one single place in the back-office. That variable will be obviously available in all the Liquid templates.

Besides, we will improve the UX of a lot of pages. For instance, end-users will be able to filter the content entries through fields defined as "searchable" by the developers.

But, our main work on Station will be dedicated to the editing experience. First, we will add a markdown editor in order to edit the editable_text elements and the text custom fields of content entries.
We also found out that it was hard for us to maintain 2 kind of WYSIWYG tools in the back-office: AlohaEditor for the inline-editing and tinyMCE in the other sections.
So, we've been looking at other editing solutions and it looks like CKEditor is the best WYSIWYG candidate. Each site will come with its own CKEditor styles so that end-users will be able to apply custom styles (CSS rules) to their content.
About the inline editing, content entries will be editable directly from the site.

On word about the API. We don't know yet if it will be part of Station or extracted as a stand-alone component.

c/ Data layer

At the very beginning of the project before Wagon even exists, we chose MongoDB as our data storage and we never looked back until now.
Engine, our new rendering component will have to deal with both MongoDB when used from Station and also directly with the filesystem in Wagon.
And if we go further, why not having Postgresql as the data storage too. Okay, you get the picture.

To achieve that, we need to think outside the Rails way a little bit. That's why we read a lot about trendy / Java stuff like DSR, DCI, Services, ...etc. To be honest, this remains a little bit blurry to us but that's the way we are going to follow for sure.