Headless CMS as a Microservice

If you love developing and blogging as much as we do, then a Headless CMS is just the thing for you. Reac
learn more about these awesome platforms!

by Wojciech Suwała · by Robert Witkowski · Jan. 10, 19 · Microservices Zone · Analysis

As a Software House that develops a lot of B2C and B2B systems, we have to deal with content management systems on a dai
We use the most popular CMS/DXP solutions like LifeRay or Sitecore. These products offer great functionalities and experien
content creators and marketing. But they also heavily affect our systems architecture and usually tightly couple our solution
chosen CMS. Also, our customers have to deal with this kind of heavy and strong dependency.

Do we have any viable alternatives? With the rising popularity of microservices and API-first approaches comes a new and p
solution – Headless CMS.

In this article, we will guide you through the process of integrating two open source CMS solutions with our sample Micronau
microservice sales portal, with a Single Page Application client written in Vue.js.

What Is a Headless CMS?

A Headless CMS is a backend only content management system which works as a content repository and gives access to this c
REST (or GraphQL) services.

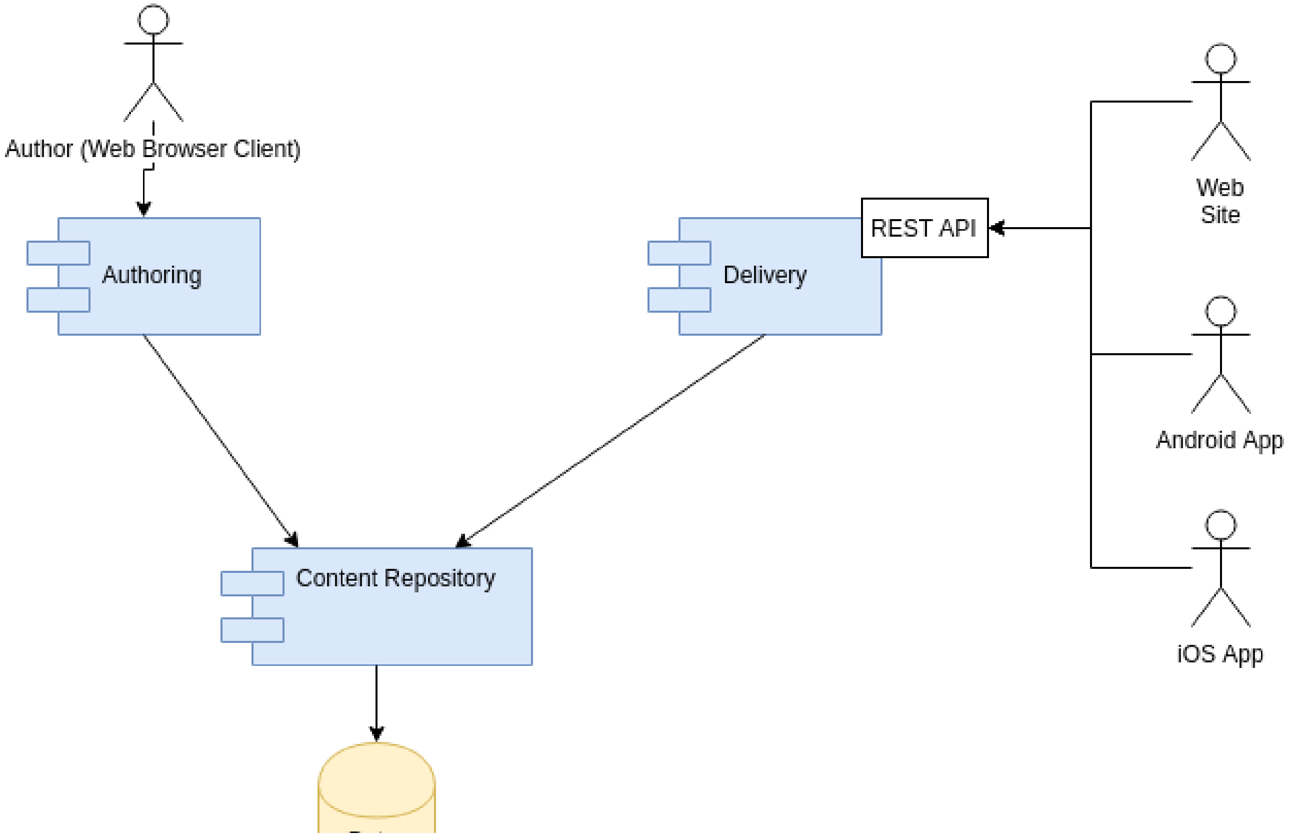
In traditional CMS, you have the following core subsystems:

- Content delivery.
- Analysis and monitoring.

Headless CMS focuses just on content creation and publication workflow. It is your application's responsibility to get the content and display it in appropriate way based on your users' needs, devices they use, and channels they operate on.

You can checkout a list of popular open and closed source Headless CMS solutions here.

Typical Headless CMS Architecture



The typical headless CMS consists of the following components:

- **Content Repository** - has the sole purpose of storing content and providing CRUD and search functionalities.
- **Authoring** - a web based application that allows authors to create and modify content, create new content types, collaborate and organize publication workflow.
- **Delivery** - uses REST APIs to expose data to an application that gives external entities access to content published and stored in the Content Repository.

Pros and Cons of Headless CMS

Advantages of Headless CMS:

- No tight coupling between business applications and CMS, resulting in flexibility that allows you to choose whatever technology and framework you like for your application.
- Headless CMS are usually much easier to deploy and use.
- Ability to easily integrate new channels, as we are not blocked by the functionalities available in a CMS.
- Nicely fits into microservice-based solution landscapes.
- Improved scalability and security due to dividing responsibilities of authoring and delivery, delivery can be separately scaled, and the authoring part can be completely hidden and not accessible to the outside world behind company firewalls.

Disadvantages of Headless CMS:

- Content authors are not able to preview how created content will look in the applications from the inside CMS.
- The analytical capabilities and content personalization features of a full blown CMS cannot be used and must be developed

Business Case: Integration of Insurance Sales Portal With a CMS

We have a simple Sales Insurance Portal which lets insurance agents select a desired product for their customers, create an offer, and set sale a policy. This system also allows for the searching and viewing of offers and policies so that they can be managed by an agent.

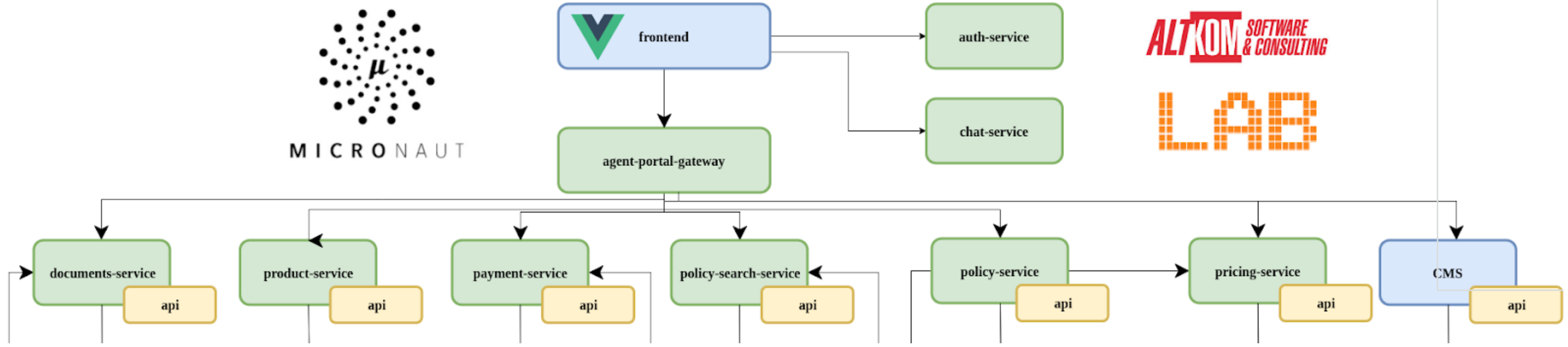
We are going to extend this system functionality by adding two new features:

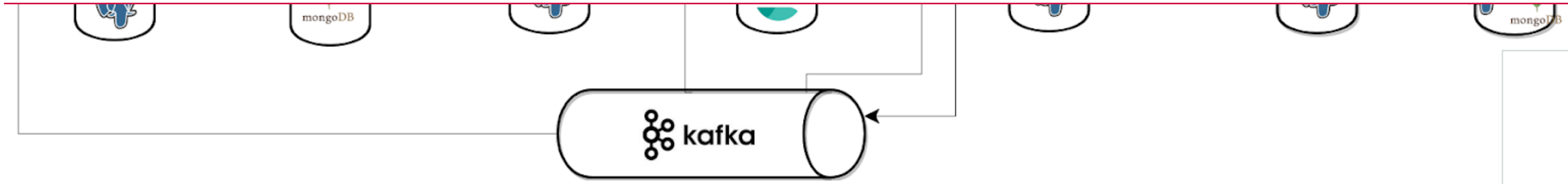
- A blog module.
- A products home pages.

The blog is going to be managed bya aCMS. Backend CMS users will create and publish posts. Each post is going to have a title, associated categories, and content. Blog posts will be accessed via REST APIs and displayed in the portal. Agents will also be able to search blogs.

The products home page will display marketing product information in a nice Bootstrap Carousel component. For each product, a description, title, and picture will be managed in the CMS and accessed via REST in order to be displayed in the portal.

Our solution architecture will look like this:





The CMS will be deployed as a backend system and will be used by content authors to create blog posts and product marketing information. Content authors will use a web application that is part of the CMS system to create and edit content.

CMS will expose REST APIs that will give access to blog posts, products, related images, and documents.

Our client web application won't access the CMS directly. We will use an API Gateway pattern for this purpose. There are many reasons for this approach. The first is that we do not want to expose CMS APIs to everybody. The second is that we want to control and centralize API access from a web client app to internal microservices and we want to treat the CMS like just another microservice. This also gives us a chance to aggregate and adjust content to the channel to which the API Gateway is dedicated. For example, we can take some data from a CMS, like product marketing info, and take prices from a product catalog microservice and combine them together.

The client application will treat the CMS like just another microservice and will access it through API Gateway REST endpoints.

This example can of course be extended. Using the same approach as outlined below we can add:

- FAQ Lists with FAQ Items.
- News.
- Event calendars.
- And much, much more.

Hippo CMS integration

We decided to give it a try because it is open source, written in Java (so we can have all our components deployed on the JVM), and customizations are possible by adding Java code. It is also a mature solution with a large user base and customer base.

Instruction with technical details (installation, configuration, exposing CMS REST APIs, accessing CMS REST APIs from an SPA app) is available here.

Full source code is available on GitHub, here: cms-integration-hippo.

Strapi CMS Integration

StrapiCMS is a typical Headless CMS for building customizable APIs.

We decided to give it a try because it is open source (like Hippo) and will always be free. And, thanks to being built on top of Node.js, Strapi delivers great performance and easy installation (through npm packages). It also has an elegant and customizable admin panel.

Instruction with technical details is available here.

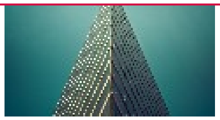
Full source code is available on GitHub, here: cms-integration-strapi.

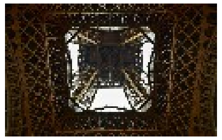
Summary

We hope that this article showed you that integration between CMS and your business application is pretty easy. With just few simple steps, you can use and display content from a CMS in any kind of application – it can be a modern web app built using Vue, React, or Angular, a mobile app developed with Xamarin Forms, or any other technology.

You don’t have to couple your application with heavy-weight CMSs just to have some basic features like blogs, FAQs, about pages, or display marketing information. You don’t have to build into your own system content editors or publishing workflows – you can use headless CMS for this.

DZone Article
Minimalistic CMS for Java With Cricket


DZone Article
Utilizing Microservices for Performance


DZone Article
Three Popular Frameworks to Build Microservices

Free DZone Refcard
Designing Microservices With Cassandra

Topics: HEADLESS CMS , MICROSERVICE ARCHITECTURE , MICROSERVICES , MICROSERVICES ARCHITECTURE JAVA

Published at DZone with permission of Wojciech Suwała . See the original article here.
Opinions expressed by DZone contributors are their own.

ABOUT US
About DZone
Send feedback
Careers

CONTRIBUTE ON DZONE
MVB Program
Zone Leader Program
Become a Contributor
Visit the Writers' Zone

LEGAL
Terms of Service
Privacy Policy

ADVERTISE
Developer Marketing Blog
Advertise with DZone
+1 (919) 238-7100

CONTACT US
600 Park Offices Drive
Suite 150
Research Triangle Park, NC
27709
support@dzone.com
+1 (919) 678-0300

Let's be friends:

DZone.com is powered by