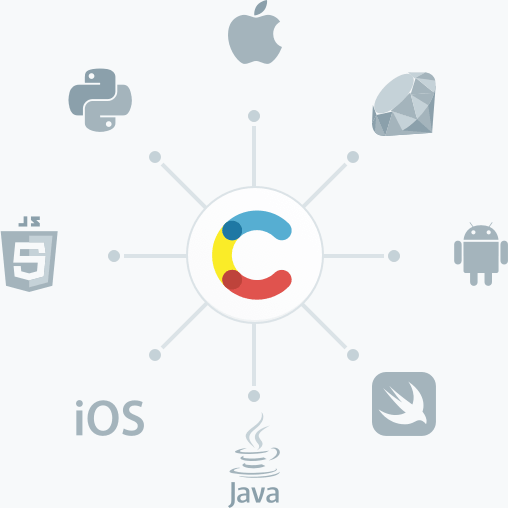
Pre sales analysis

Contentful

Rahul Yadav



# Contentful

Three core concepts of the contentful that picking up the market.

* **Fast**: Now you don’t wait for setting up the bulky environment and consumes development time in just installation and setting up the environment. This approach contributes in faster implementation and faster performance and scalable content migration and transformation using code.
  + **Faster implementation**: It’s a cloud-native SAAS with CDN, you can create, manage your data just accessing to the CDN.
  + **Faster performance**: Data is separated from view, provided faster user experience, and flexibility of choosing any development tools make it easier to adapt technology which provide faster user experience.
* **Flexible**:
* **Modern Architecture**

Contentful is the essential API-first content management infrastructure to create, manage and distribute content to any platform or device. Which can do the belo

* **Launch digital products faster**: Content infrastructure helps digital teams work better together by unifying content in one hub, so marketers and developers can build simultaneously.
* **Create compelling experiences:** With the freedom to use their favorite tools and frameworks, developers can focus on building leading-edge websites, apps and omnichannel experiences.
* **Store and edit content in one hub:** Editors have full control of content in an intuitive web app. Marketers can test and refine, publishing changes to every platform from one interface.

## Different approaches of site development

* **Traditional CMS Architecture**: the traditional CMS architecture tightly links the back end to the front end. Content is created, managed, and stored—along with all digital assets—on the site’s back end. The back end is also where website design and customization applications are stored. This content management back end and database is bound within the same system that delivers and presents content to devices and end users (front end).

So, with a traditional CMS, your editors are writing and publishing on the back end of the same system your website visitors are viewing. Blogging platforms, such as WordPress, Squarespace, and Wix, are examples of traditional CMS architecture.

To recap, a traditional CMS is comprised of:

* + A database where content and digital assets are stored (back end)
  + A content management back end where content is created (back end)
  + An application where publishers and designers create and apply design schemas (back end)
  + A front end that displays published content on HTML page.

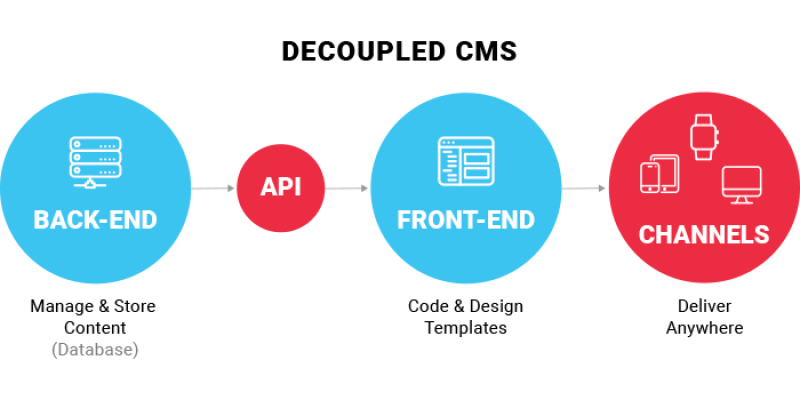


* **Decoupled CMS Architecture**: Decoupled CMS architecture separates—or decouples—the back-end and front-end management of a website into two different systems: one for content creation and storage, and another system, one or more, are responsible for consuming that data and presenting it to the user through some interface. In a decoupled CMS, these two systems are housed separately. Once content is created and edited in the back end, this front-end agnostic approach takes advantage of flexible and fast web services and APIs to deliver the raw content to any front-end design on any device or channel.

Even though the back end and the front-end application function independently of one another, the front-end architecture is predetermined with a specified delivery environment (for example, React or React Native). Thus, the two systems are tightly linked and can function as one.

From a technical standpoint, a decoupled CMS is comprised of:

* + A database where content and digital assets are stored (back end)
  + A content management back end where content is created (back end)
  + An API that connects the content management back end with the front end
  + A default content publishing front end



* **Headless CMS Architecture**: What is a headless CMS? Well when defining headless CMS architecture, it’s important to understand how it relates to decoupled. Headless architecture is actually a subset of decoupled. Both have a content management and storage back ends and deliver content from that database through a web service or API. But the key difference is the presentation layer: unlike decoupled, headless does not have a defined front-end system or presentation environment.

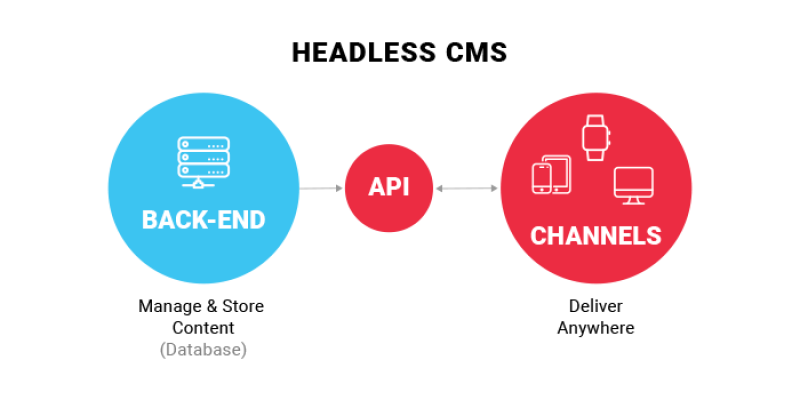
An easy way to understand the difference is to think of decoupled as proactive and headless as reactive. Decoupled prepares the content on the back end and then can proactively deliver and present formatted content to various channels. Headless, on the other hand, is a content-only data source and has no functionality within the CMS to present content to an end user on its own. Content is created and managed, but it just sits there, available and waiting to be called upon by an API and delivered to applications and systems.

This means that headless is “API only, UI anything”; it can push content to any device or channel with internet access. It can publish the same content to a website, an app, a wearable device or any device connected via Internet of Things (IoT) because the content isn’t bound by a predetermined user interface.

Of course, “headless” doesn’t stay headless assuming, that is, that you want to actually publish the content you create and manage. Something must serve as the “head” for content presentation– but it’s simply not attached, by default.

From a technical standpoint, a headless CMS is comprised of:

* + A database where content and digital assets are stored (back end)
  + A content management back end where content is created (back end)
  + An API that connects the content management back end to any device or channel
  + The ability to connect to any publishing front end, allowing organizations to have the front-end technology of their choosing



## Problem with traditional/Decoupled approach

Each CMS architecture has its pros and cons and is more appropriate in specific environments than others. There is no one superior approach and the architecture that will work best for you depends on the needs of your business. Here is our breakdown of the pros and cons of the three architectures.

* **Traditional CMS**: Traditional architecture is great for simpler sites, it isn’t ideal for those that are more complex. A traditional CMS restricts the type of content (i.e. video, audio, and advanced imagery) editors can publish and where that content can appear. Since the front end and back end are locked together, the programming framework can quickly become limited and for developers this means customization is slower and less agile. The front-end and back-end interdependence also translates to more time and money required for maintenance and enhancements.
* **Decoupled CMS Architecture**: A decoupled CMS is more complicated than traditional and does require extra development work compared to headless—especially around building the front end.

## How headless is better approach for the market

Headless architecture doesn’t include any frontend system whatsoever. Instead, content is published to an API or web service that is capable of pushing content straight to any smart device.

A major benefit of using a headless CMS is that the same content can be published to a website, an app or anything connected to the internet of things. In the long run, the headless approach has practical implications for the IoT and artificial intelligence; in the short run, it can make managing content across different delivery formats much easier since the content isn’t bound to a predetermined structure. A headless CMS has only:

A content management backend

* An API
* The Be

Benefits of Headless CMS Architecture

As we’ve already mentioned, there are a number of benefits to using a headless CMS over a traditional or coupled CMS. The top five benefits include:

* **Flexibility**: Some developers find traditional CMS architecture to be frustratingly limiting. Using a headless CMS gives you the freedom to build a frontend framework that makes sense for your project. Since every headless CMS comes with a well-defined API, developers can spend more time focusing on content creation rather than content management.
* **Faster time-to-market**: Speaking of which, creators needn’t concern themselves with how different frontends display their content since all updates are pushed across all digital properties. This not only speeds up production, but it also allows you to reuse and combine individual content items.
* **Compatibility**: You can display content to any device while controlling the user experience from one convenient backend.
* **Extra Security**: Since the content publishing platform isn’t accessible from the CMS database, using a headless CMS lowers your risk of DDoS attacks.

## Use Cases for Headless

Headless content management can be ideal for the following use cases:

* Websites and apps utilizing JavaScript frameworks like VueJS, React or AngularJS
* Websites created with a static site generator
* Any ecosystem where the same content is published across multiple delivery platforms

If you feel like a traditional CMS imposes too much rigidity onto your project, then a headless option may help you more accurately achieve your vision. Mobile developers especially benefit from headless content management since the API allows them to deliver content to an iOS or Android app from the same backend.

The headless approach certainly allows developers to create more engaging user experiences, but it’s not necessary for small company websites with only a few pages. For simpler projects, using a traditional CMS like WordPress is the faster and more efficient way to go.

## Headless CMS Options

Now that we’ve covered what a headless CMS can do, let’s look at some of the tools developers have to choose from:

* **Directus:** Directus is great for projects that require highly customized databases. Written with Backbone.js, this open source CMS boasts many features including asset management, messaging, commenting, versioning and internal extension points. The public API uses REST.

Like many traditional open source CMS tools, Directus is powered by PHP, so developers familiar with the scripting language should feel right at home. Businesses can benefit from the hosted option if they don’t want to be bothered with scalability or maintenance, but WordPress veterans should have an easy enough time with the free version.

* **Cockpit CMS:** Cockpit gives developers more control than a traditional CMS by allowing them to use the frameworks and programming languages of their choice while authorized individuals maintain full control over content management. Limiting when and where updates can be made helps avoid unintentional changes to settings and plugins, so a small mistake will never render an entire website broken. Best of all, this headless CMS is open source.

Cockpit CMS is the product of a single developer, so support is lacking and updates are rare. While other developers can add code to the project, keep in mind that no one may be reviewing or correcting it for some time. Security features are also lacking compared to other headless CMS software. If you’re working on a small project, and you just want a little extra flexibility, Cockpit is a good way for developers to get their feet wet.

* **Contentful:** Contentful is welcoming to newcomers, so anyone can start creating data entries relatively quickly; however, as you delve further into its features, you’ll find that this CMS has quite a bit of depth. It’s also one of the more affordable options for small or medium sized businesses.

While APIs are available to help with content display and editing, developers are completely responsible for functionality. If you’re used to working only with coupled architecture, you’ll have to get comfortable with concepts like content modeling and search-based management. Nonetheless, overcoming these learning curves can be well worth the endeavor.

## **Case study of resources.**

**3 Core concept.**

* **Fast**:
  + Ease access through cdn as cloud-native SAAS
  + New products can launch faster comparatively to the market.
  + Use the development tools you know.
  + Fast, scalable content migration and transformation using code.
* **Flexible:** 
  + Build any user experience and for any device (anywhere).
  + Flexibility in choosing developer with different language domain.
  + Ease localization
    - Large set of digital products covered
      * Chatbot with product recommendation
      * Kiosks, Display, menus
      * Mobile apps
      * Audio guides, onsite-displays, immersive website.
* **Modern architecture**
  + Modern stack architecture micro-service architecture to combined with others
    - Algolia search
    - Segment
    - Optimize
    - AWS
    - Azure
    - Google cloud platform
  + Built-in integrations with popular platform.
  + Connect platform easily using webhooks and user extensions

## **Good fit use cases**

1. **Speed and Velocity**
   1. **Challenges:**
      1. Build website fast
      2. Tight deadline
      3. Both developer and editor to move parallel and fast
      4. Need light weight product.

**Keywords:**

Modern stack website, mobile application, micro-site, landing page, multi-campaign initiatives.

1. **Enabling content operations at scale:**
   1. **Challenges:**
      1. Modernize thousands of websites organization wide organization wide and Global consistency
      2. Central governance and local velocity in different markets
      3. Modernize global architecture for today and future

**Keywords:** Content Hub, Central shared service layer for landing pages, blogs, microsites, Shared architecture / content models

1. **Building engaging, cutting edge products:**
   1. **Challenges:**
      1. We need to build engaging and modern applications that show off our brand
      2. Add multiple services together to personalize experience at scale.
      3. Data optimization on every step of journey

**Keywords:** Modern applications/websites or portfolio of websites that need to be personalized, Mobile, AR/VR applications, Digital signage, dynamic digital menus, etc.

**Keywords:** Digital Transformation,

1. **Enabling the digital transformation journey:**
   1. Challenges