

**Acquia**  
EXPERIENCE DIGITAL FREEDOM

# ***HEADLESS AND HYBRID DRUPAL 101***

How to determine which CMS approach will work best  
for your organization



# TABLE OF CONTENTS

**03**

**INTRODUCTION ▶**

**07**

**HOW A UNIFIED ARCHITECTURE WORKS ▶**

**09**

**HOW A DECOUPLED ARCHITECTURE WORKS ▶**

**11**

**THE DIFFERENCE BETWEEN HEADLESS AND DECOUPLED ▶**

**13**

**OPTIONS FOR BACK-END WEB SERVICES ▶**

**16**

**OPTIONS FOR FRONT-END SDKS ▶**

**19**

**GO BEYOND API-ONLY ▶**

**22**

**THE ONE QUESTION THAT REVEALS THE RIGHT APPROACH ▶**

**27**

**FLEXIBILITY FROM A HYBRID ARCHITECTURE ▶**

**30**

**THE BENEFITS OF PROGRESSIVE DECOUPLING ▶**

**33**

**DRUPAL LETS YOU CHOOSE THE RIGHT APPROACH FOR EACH PROJECT ▶**

# INTRODUCTION

**In the web development world, few trends have spread more rapidly than headless content management systems (CMS) and decoupled applications.**

Clearly, it's an evolution beyond the traditional approach, where the presentation layer was tightly coupled within the CMS on the back end. But having more options also drives a need to better understand how each works so you can choose the approach that best aligns with your goals.

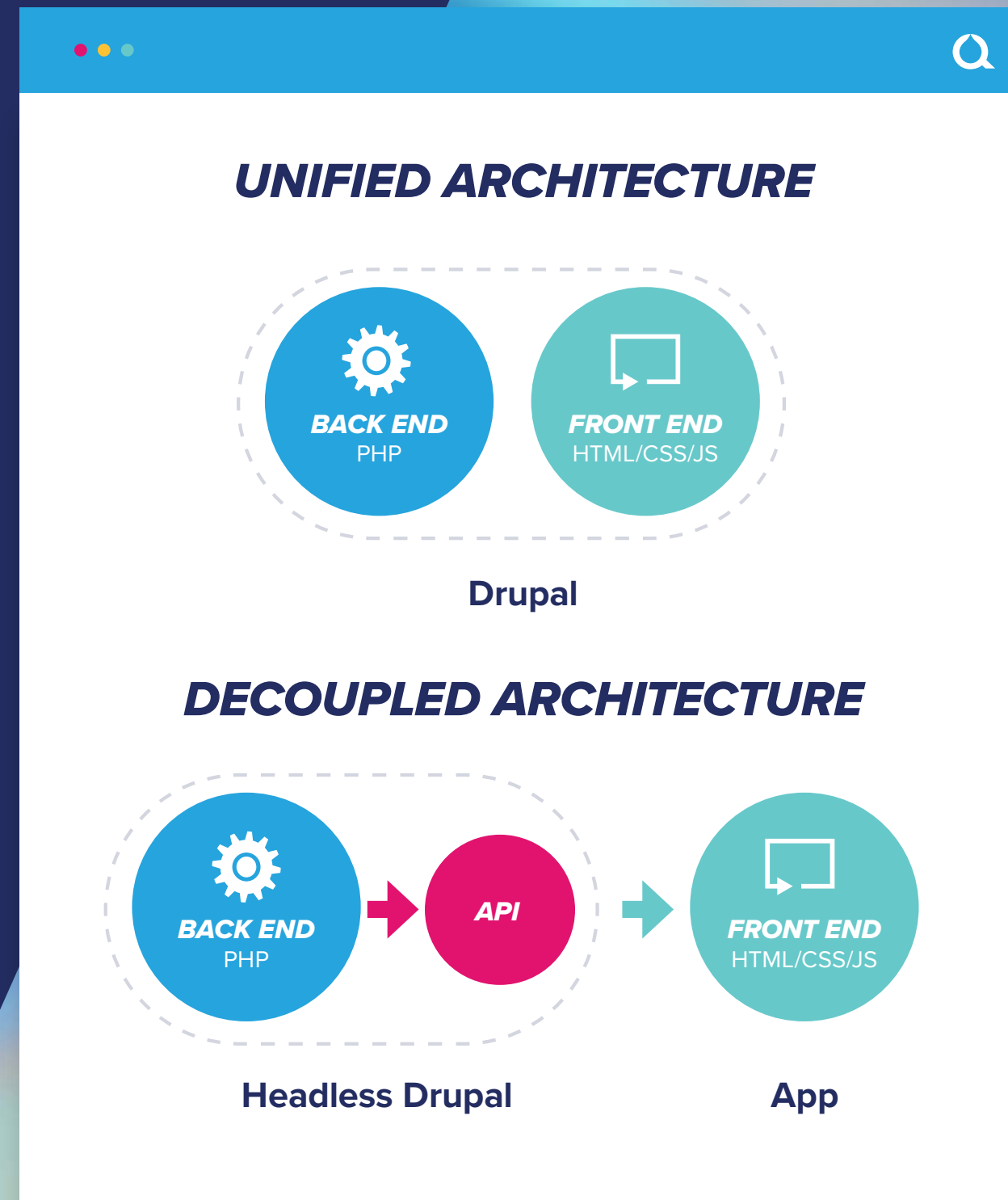
In this e-book, we explain how a decoupled architecture works, when you should consider decoupling, and how both marketers and developers can leverage headless Drupal to deliver ambitious digital experiences. We'll also cover how Drupal can provide advanced "hybrid headless" capabilities that are difficult to find in other systems.

## Let's begin by clarifying how “unified” and “decoupled” architectures differ.

In a traditional or **unified architecture**, both front-end and back-end responsibilities are contained within a single system.

This allows the CMS to not only manage content, but also render the markup (HTML) for the front-end experience using templates or low-code tools. This packaging of tools provides a single, **unified** system suitable for a wide variety of websites and applications without requiring additional complexity.

With a **decoupled architecture**, the Drupal back end provides an API service layer for serving structured content. Here, rather than use the Twig templating engine that comes out-of-the-box, a developer would leverage another technology to render the front-end experience. When you think of the growing number of devices that allow users to gather and interact with content (through “the glass”) this makes a lot of sense.



In this model, Drupal is considered a headless CMS repository, which exposes content and data for consumption by other applications. These applications could include:

- /// **Native applications** are developed for one specific device or platform, such as desktop or mobile. For example, mobile applications are often developed for particular smartphones or smartwatches.
- /// **JavaScript applications** update pages dynamically through asynchronous requests back to the server and eschew full page refreshes. This means that web applications can call the server without the browser needing to reload the page.
- /// **Digital signs** are common ways of displaying information in public spaces, restaurants, offices, or other locations. Typically, these are digital displays that can request content and images to display from an external API, allowing those signs to easily be updated and managed remotely.
- /// **Internet of Things (IoT) applications**, which include devices like smart TVs, the Amazon Echo, Apple Watch, or connected fitness trackers. This space is rapidly expanding, and these devices often rely on external services for content and data.

## ***WHY COMPANIES MIGHT GO WITH A HEADLESS APPROACH***

Organizations may select a headless Drupal approach for a few reasons. Some implement a headless strategy to leverage Drupal as a content repository to serve content to any device in a complex digital ecosystem. Others favor decoupling to allow front-end teams to use popular JavaScript (JS) frameworks while maintaining the back-end capabilities of Drupal. For example, many organizations leverage headless Drupal in combination with JavaScript frameworks, such as React, Svelte, NextJS, or VueJS.

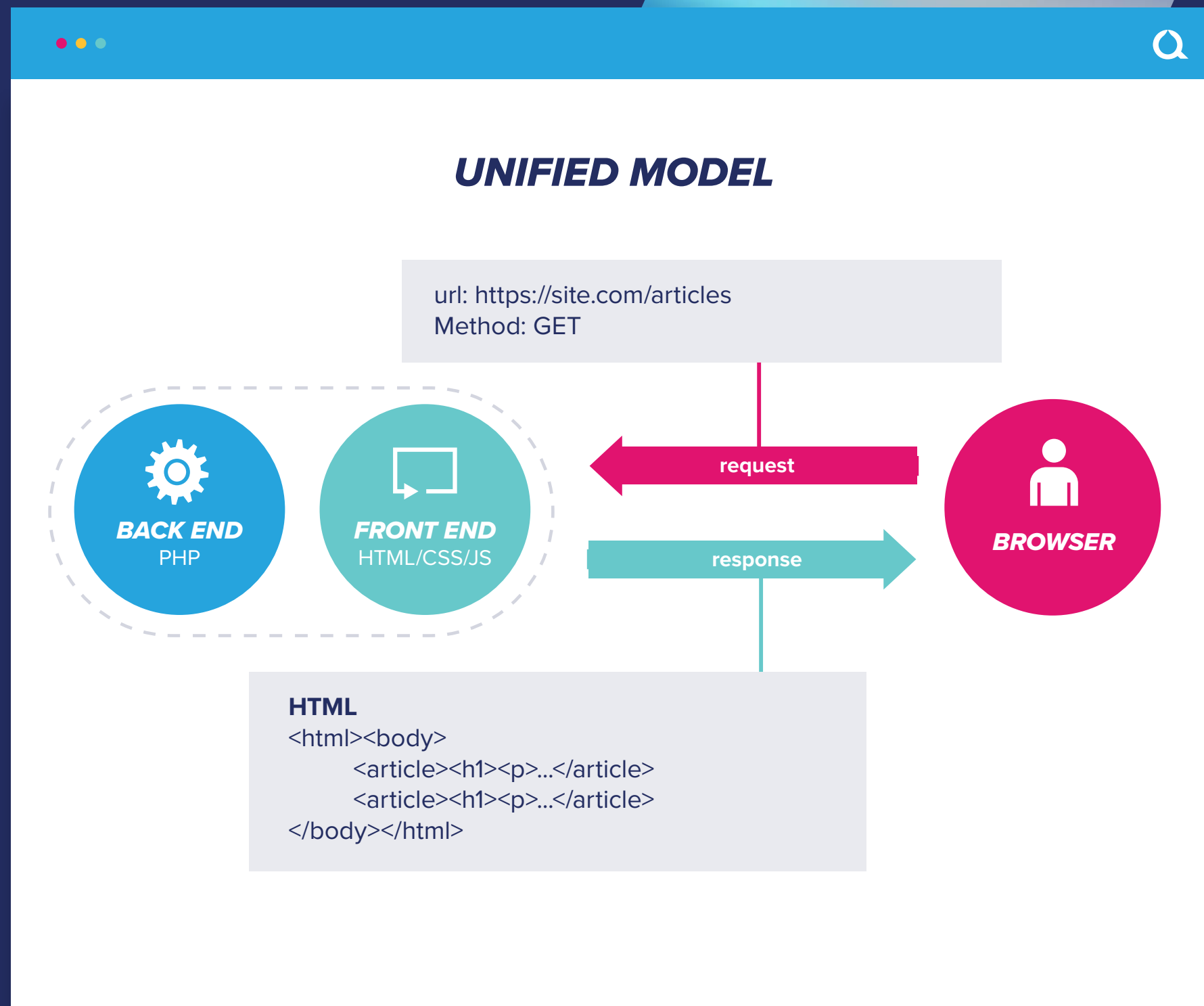
One important consideration to factor here is that no matter what's driving the decision to go with a headless CMS, it pays to make sure both web developers and marketers get what they need from it.

# ***HOW A UNIFIED ARCHITECTURE WORKS***

Most interactions that take place on the web rely on a **request/response** paradigm where a user requests data and a system responds by gathering, formatting, and rendering the appropriate content within a predefined HTML template.

This is the traditional way the internet works: the browser makes a request for HTML, the server returns the data, and then the browser renders the page to the user. Very simple, very scalable, and truly foundational to how the web works.

In a unified model, Drupal provides the back-end content management system and an HTML rendering engine (Twig) in a composable framework. While some traditional CMS systems provide similar results, the difference with Drupal is that it is **API-first** and based on discrete, composable systems. This provides the opportunity to use it in either a unified or a decoupled architecture depending on your needs.





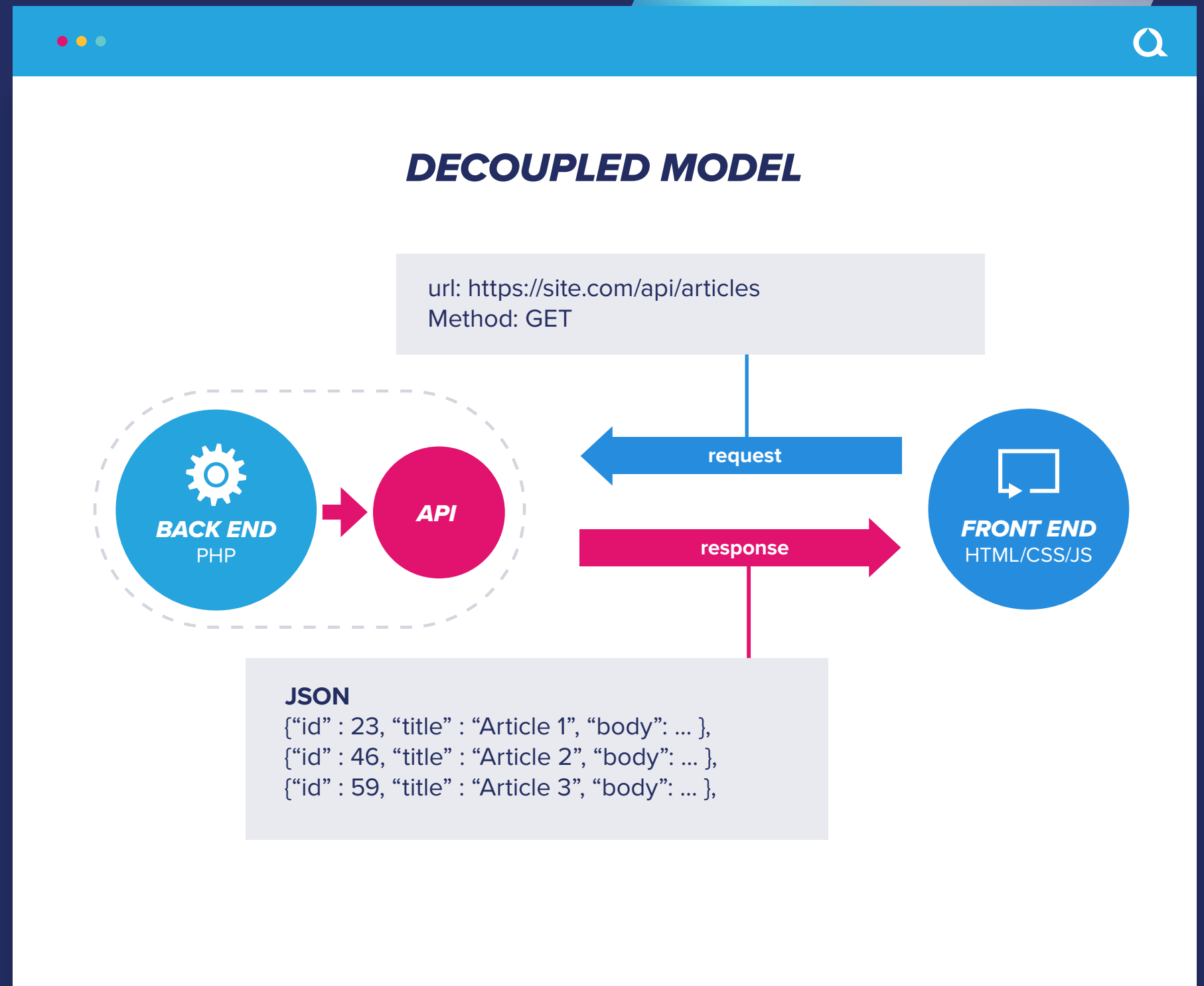
# ***HOW A DECOUPLED ARCHITECTURE WORKS***

A decoupled architecture takes the same basic approach, but with a slight difference. Instead of all of this taking place within a single, unified system, it distributes these responsibilities among multiple systems.

In a decoupled architecture, the Drupal back end acts as the content repository. In this scenario, the headless Drupal repository and the decoupled application exchange data through standard HTTP methods. The application makes a request and passes parameters to the API. Then the headless Drupal CMS returns the response, which is typically in JSON format.

Drupal is API-first, and the RESTful Web Services module is included in Drupal core (the standard base package that defines the latest release of Drupal). The module provides a customizable, extensible RESTful API of data managed by Drupal. An HTTP response can be served in JSON, XML, or other representations.

In this figure, the REST API and HTTP client act as the mediators in the decoupled architecture, allowing both back-end and front-end developers to work with their preferred frameworks. Drupal provides basic REST APIs and a fully functional JSON:API format in core, with GraphQL and other formats available in the ecosystem.





***THE DIFFERENCE  
BETWEEN HEADLESS  
AND DECOUPLED***

In this guide, we have chosen to use the terms **headless** and **decoupled** very specifically. The distinctions between headless and decoupled can be pretty nuanced and difficult to define.

A good rule of thumb is this: “headless” typically refers to the CMS or the service providing data via the API, and “decoupled” typically refers to the architecture or the front-end application itself.

Simply put, a **headless CMS** does not provide any display to the end user, and only provides the API to make content and data available. This allows any front-end experience to use the content. Drupal is a powerful and popular choice for a headless CMS because it is API-first (not API-only). This means that it was specifically designed with an API services layer at its core.

A **decoupled architecture** typically refers to some type of application that provides the front-end experience, along with one or more API services to provide the content and data. So, we would typically include the user interface in the decoupled architecture, whether that is a JS application, mobile app, smart TV, digital signage, or something else.

## **A HEADLESS DRUPAL ARCHITECTURE ALLOWS WEB DEVELOPMENT TEAMS TO:**

- Power a multitude of devices:** With its flexible APIs and web services, Drupal can be the brain behind all of your systems to deliver content everywhere.
- Leverage other front-end technologies:** Drupal can function as a services layer to allow content created in the Drupal CMS to be presented through a JavaScript framework, such as React, VueJS, and Angular.
- Control all aspects of your media:** Headless Drupal can serve as a central repository to send video and data to the many outlets available in the current media marketplace.
- Integrate with multiple systems:** Organizations can introduce Drupal to the back end in order to support existing technical systems.

# ***OPTIONS FOR BACK-END WEB SERVICES***





Today, Drupal offers a range of web services modules, which enable anyone to consume data from Drupal with ease. These include community modules in addition to modules that are now available in core.

**/ Core RESTful Web Services:** Drupal offers a REST API out of the box. This includes operations to create, read, update, and delete (CRUD) content entities and to read configuration entities. There are also four primary REST modules in core, including **Serialization**, **RESTful Web Services**, **HAL**, and **Basic Auth**. Core REST requires limited configuration while providing a wide range of features.

**/ Core JSON:API: JSON:API** is increasing in popularity due to its adoption by developers as a common format and its robust support for complex data. JSON:API is a specification for REST APIs using the JSON format and offers functionality beyond the core services layer.

/// **GraphQL:** Originally developed by Facebook, GraphQL is a query language developed for client-tailored queries. GraphQL enables clients of headless Drupal to easily pull data from the back end, allowing them to retrieve custom sets of data through a single request. Drupal supports the **GraphQL module**.

/// **Low-Code Query Builder:** Drupal provides a core module called **Views** that provides a UI to build components that fetch content from the database of your site and present it to the user. This dynamic query builder can provide REST endpoints that are completely managed in the UI. This is a powerful way to create and manage custom endpoints without writing code.

/// **Custom Code:** The service layer in Drupal is designed to be pluggable and composable from the start. This means that even if you have some very unique or even proprietary service integrations to provide, you can build what you need on top of Drupal.

This customization is going to be much faster, easier, and more stable than almost any other bespoke approach because you can simply extend the CMS to provide what you need instead of starting from scratch.





# ***OPTIONS FOR FRONT-END SDKS***



In addition to a robust collection of web services, Drupal now offers an ecosystem of software development kits (SDKs), which help to accelerate the development of applications in other technologies. SDKs can help to extend Drupal's reach outside of PHP.

Previously, consuming Drupal content required some understanding of the REST API implementation details and custom application development. As a truly composable CMS, the Drupal community has continued to provide support for popular frameworks and application development platforms.

This means that you can get a working application on day one with a multitude of options:

▸ **Drupal State** offers a common set of utilities that allow JavaScript developers with limited knowledge of Drupal or the JSON:API spec to take advantage of the best features of Drupal's API.

Being framework-agnostic, Drupal State can be used with vanilla JavaScript or any other common JavaScript frameworks. Drupal State aims to be modular, extensible and overridable, helping support the Drupal JavaScript ecosystem.



**/ Next.js for Drupal**, or Next-Drupal, provides a path to replace your Drupal front end with Next.js while retaining key Drupal editing features — the best of both worlds. This includes the ability to instantly preview the Next.js site within the Drupal editorial workflow, to create custom content architecture without the hassle of complex modules and clunky Twig templates, and to support Multisite and Multiple Views.

**/ Gatsby**, a static-site generator built with React and GraphQL, alleviates the pain points around scalability and performance associated with building React applications. On the back end, by leveraging Drupal's content modeling, creation and editing tools along with the JSON:API module to serve content to

the Gatsby front end, you get a powerful, full-featured, open source, and free alternative to expensive enterprise content management systems.

**/ Druxt.js** is an open source bridge between two frameworks, NuxtJS and Drupal, with the ability to leverage Drupal's own Entity/Field display system, Block regions, Views and more. DruxtJS supports the Drupal JSON:API client with Vuex caching and acts like a Vue.js theme layer for Drupal. Druxt components can be themed using Wrapper components alongside Vue.js slots, \$attrs, and props.

# ***GO BEYOND API-ONLY***



Today, common headless CMS platforms take an **API-only** approach. Utilizing the decoupled architecture, organizations simply get an API and a back end to input data into the CMS. The required front-end experience is missing.

The code they would need to build a complete experience, whether a website, an app, or other omnichannel experiences, must be created. This is why the decoupled architecture is often referred to as “high code;” It relies on developers to write and maintain code for the front-end experience. A unified architecture is often termed “low code” because it provides UI tools for non-developers to assemble the experience.

For some applications, the high-code approach is acceptable and even desired. There are specific times when the experience and the code should be tied together, like when you are building a mobile, web, or IoT application.

When the developer owns the experience, this is often the most efficient way for them to build and maintain the front end.

However, this is a severe limitation when you have different needs, like low-code tools to empower business users. In this model, any kind of custom layout requires a developer to implement, which typically slows down the time to market. To make matters worse, API-only headless CMS platforms tend to push organizations

into more expensive tiers by placing limits on your site architecture and governance.

For instance, you could be constrained by the number of content types and fields, the number of users, or even the ability to have custom roles and permissions. As your usage of their platforms matures, it can get more expensive.

Alternatively, Drupal utilizes an **API-first** approach where an API is always available, but is not *required*.



**There are no limits on the number of content types, complexity of data, number of integrations, number of users, or even advanced options for roles and users.**

Drupal provides all the functionality of a headless CMS platform without the limits. It also allows organizations the ability to choose which architecture is right for each project.

When necessary, the unified architecture takes it a step further by giving organizations the tools, templates and other rendering options needed to build out a site, without limitations. This is a massive advantage because **when you choose Drupal, you get the best of all worlds**. You get the ability to use the unified or decoupled architecture on the same platform as needed — and sometimes at the same time. This is the hybrid CMS approach, which provides the greatest degree of freedom and flexibility.



***THE ONE QUESTION  
THAT REVEALS THE  
RIGHT APPROACH***

There is a lot of hype around decoupled architectures, so before embarking on a project, it is important to make a balanced analysis. If you choose to use Drupal as the content repository to serve multiple applications, a decoupled architecture could be right for you. Primarily, you can determine this by asking one simple question: ***Who owns the experience assembly?***

## **DEVELOPER OWNERSHIP = HIGH-CODE ASSEMBLY**

If the experience is code-driven and owned by the IT team (developers), then a *high-code* approach using the decoupled architecture might be best. This provides a more limited tool set for the business user to streamline content creation, while the developer has the greatest power to build the experience, templates, and code to display that content. The experience and the code are managed and deployed together.

## **THERE ARE SEVERAL ADVANTAGES ASSOCIATED WITH A FULLY DECOUPLED ARCHITECTURE:**

### **/ Separation of concerns**

Utilizing Drupal strictly as a content repository can enforce a separation of concerns. With a fully decoupled architecture, the handling of content is confined to the back end. This is separated from the front end, which only addresses the presentation and delivery of that content to the end user.

### **/ Pipelined development**

This separation also extends to both back-end and front-end teams and allows developers to work at their own independent development velocities. Front-end developers are free to control markup and rendering, while back-end developers can focus their efforts on developing a robust API.

### **/ Streamlined content management**

Moving experience management responsibilities to the development team frees up business users to focus on structured content alone. With fewer responsibilities required for content creation and management, the CMS can be streamlined to provide a simpler and sometimes faster workflow for creating content. When the content is highly standardized, this efficiency can be a huge asset.

**However, when the entire front end is controlled by a decoupled application, technical teams cannot take advantage of the Drupal capabilities many users value.**

This strategy of fully decoupling negates Drupal capabilities like in-place editing and display management. It also introduces additional points of failure and the risk of increased technical debt.

Other risks include:

- / No cross-site scripting protection or input sanitization (especially when not using a framework)**
- / No layout and display management for business users**
- / No previewable content workflow**
- / No off-the-shelf modules or integrations affecting the front end**
- / No system notifications, errors, or messages**
- / No BigPipe progressive loading or advanced caching strategies**
- / No OOTB accessible markup or user experience benefits**
- / No OOTB multilingual management and language switching**





## MARKETER OWNERSHIP = LOW-CODE ASSEMBLY

On the other hand, if the experience is content-driven and owned by the marketing team (non-developers), then a *low-code* approach using the unified architecture should work best.

This provides self-service tools to empower the business user to control the presentation, while allowing the developer team to build features, decouple components, and work in parallel tracks. The experience and the code are managed separately.

There are many **advantages** associated with a unified architecture with Drupal:

### **/ Low-code tools**

One of the most powerful and noticeable advantages is the ability for non-developers to control the assembly and creation of content using low-code tools. This means that they can use the drag-and-drop UI to “build” content and adapt it to meet their specific needs. Instead of requiring a developer to build templates, the business user can directly create the experience they need as quickly as possible.

### **/ Experience preview and workflow**

With low-code tools, the content creator has the ability to visually build the pages and experiences they need. This immediate feedback increases velocity and enables them to get their work ready for publication in hours instead of days. The workflow tools also mean that content approval processes can not only enforce governance, but also give the approvers the same preview options.

### **/ Ambitious digital experiences**

In addition to assembling beautiful static content, low-code tools also enable creators to assemble dynamic and advanced components. These components, provided by developers, are often the same pieces of a fully decoupled application broken into reusable, “bite-sized” pieces. This enables non-developers to create compelling, modern experiences that are functionally similar to a fully decoupled application.

These advantages are why Drupal is such a well-loved CMS by many marketing teams. However, with power comes responsibility. When adopting a low-code approach, more thought and effort needs to be devoted to governance and management of the content creation process itself. The ease with which the system can be extended and modified can make it too easy to grow.

It can become a challenge to manage the more complex configuration options and keep the CMS from growing into more of a sprawling and inefficient system. It often requires discipline and strong technical guidance to find the balance between growth and more limited standardization.

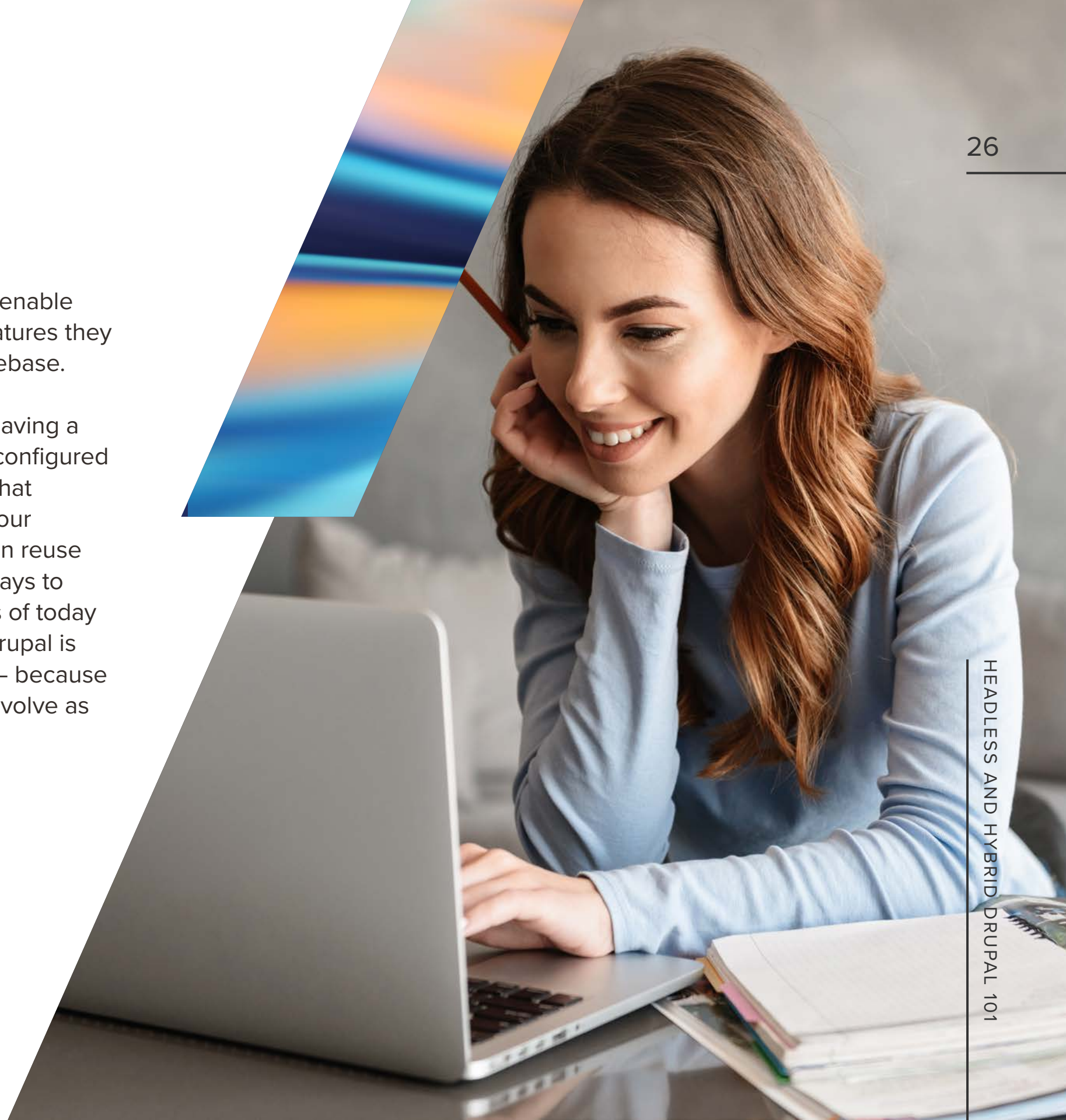
## **BOTH APPROACHES ARE VALID**

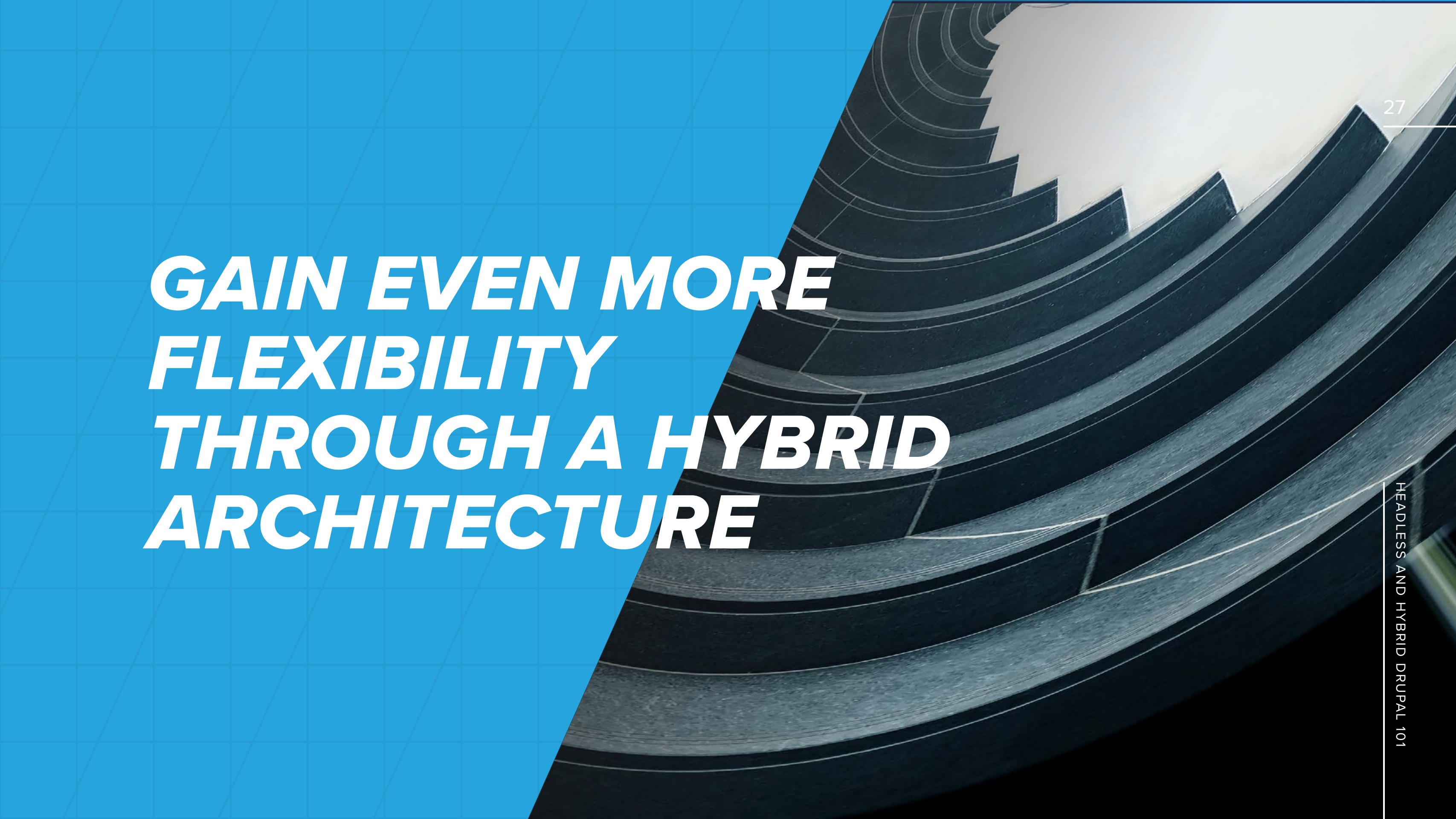
There are advantages and disadvantages to both approaches, and there is no “right” way to go. This is why it’s important to establish who will own experience assembly, how changes are delivered, if the experience needs to be separate from code deployments, and the needs of the project in question.

In any event, Drupal provides the best CMS choice for your organization because it can be used for any architecture, as opposed to API-only tools. In fact, you can actually serve multiple applications with different needs from a single codebase because Drupal natively supports multisite management and is a composable platform.

Each application can simply enable the specific modules and features they need from the common codebase.

Organizations benefit from having a standard tool that can be reconfigured in multiple ways. It ensures that you are never “stuck” with your architecture, and that you can reuse the same tools in different ways to solve the business problems of today and tomorrow. This is why Drupal is often called “future-ready” — because it is designed to be able to evolve as the industry changes.

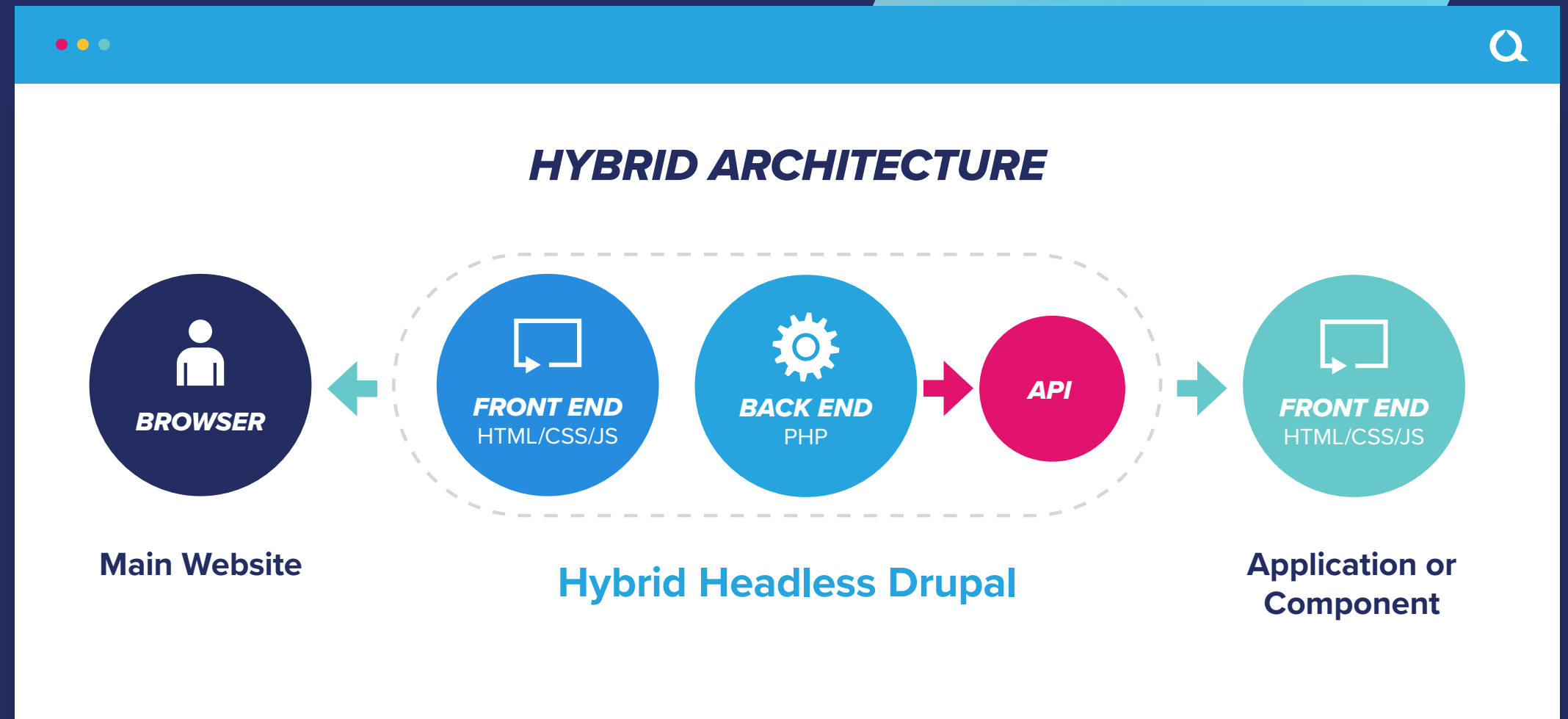




***GAIN EVEN MORE  
FLEXIBILITY  
THROUGH A HYBRID  
ARCHITECTURE***

Thus far, we have been primarily focused on unified versus decoupled architectures. Retaining a unified CMS architecture is a perfectly valid option for many sites and apps. However, if you suspect that your needs go beyond Drupal's typical offerings, you could select either a **fully headless** or **hybrid** strategy. In fact, many people think that this is a binary choice: it's either unified or not. However, there is a third option that has grown increasingly popular among Drupal users: hybrid architectures.

A core risk of a fully decoupled application is that you have to recreate and manage many capabilities and features that Drupal provides in a unified model. This can translate into higher costs, longer development cycles, and reduced support for non-developer users. This is why many organizations are now looking to leverage Drupal as a true **hybrid CMS**.



When using Drupal in a hybrid architecture, you are able to combine the benefits of a unified system with the advantages of a headless CMS. This is because Drupal is API-first. That is, the API services layer is available in Drupal core along with all of the presentation tools. This hybrid approach is often the ideal option for modern organizations that need to balance the needs of the marketing team to own the experience with the developer team being asked to implement advanced features.

Once we get into hybrid architectures, things can get more complicated because we realize that it is no longer a binary choice, but a spectrum of options and opportunities. This flexibility is one of the things that developers love most about Drupal.

## **WE TYPICALLY SEE TWO PRIMARY APPROACHES TO IMPLEMENTING A HYBRID ARCHITECTURE:**

### **1. Embedded decoupled components**

Using Drupal as a low-code assembly tool, non-developers can create content and put different components together to build the experience, which is then rendered as sent to the browser. Drupal also supports the ability to add a library of decoupled components for assembly with the other CMS assets.

These can be native web components or any other JS framework supported component. These components can provide advanced interaction options, or even be mini applications themselves, interacting with any web-based services like a full JS application. In fact, these components can also use Drupal's API layer to interact with the system as a headless CMS, or tie into external systems, such as an e-commerce checkout.

### **2. Satellite decoupled applications**

The other major approach we see is to allow Drupal to operate in a unified architecture for the main site, but then offer the content via API for consumption by "satellite" applications. These can be any of the decoupled applications we have discussed above: JS frameworks, mobile apps, smart TVs, and other IoT applications.

This flexibility is why a hybrid architecture is touted by developers and **industry analysts** alike. It provides the adaptability promised by headless services with the tools and support of a unified application.

***CONSIDER THE  
BENEFITS OF  
PROGRESSIVE  
DECOUPLING***



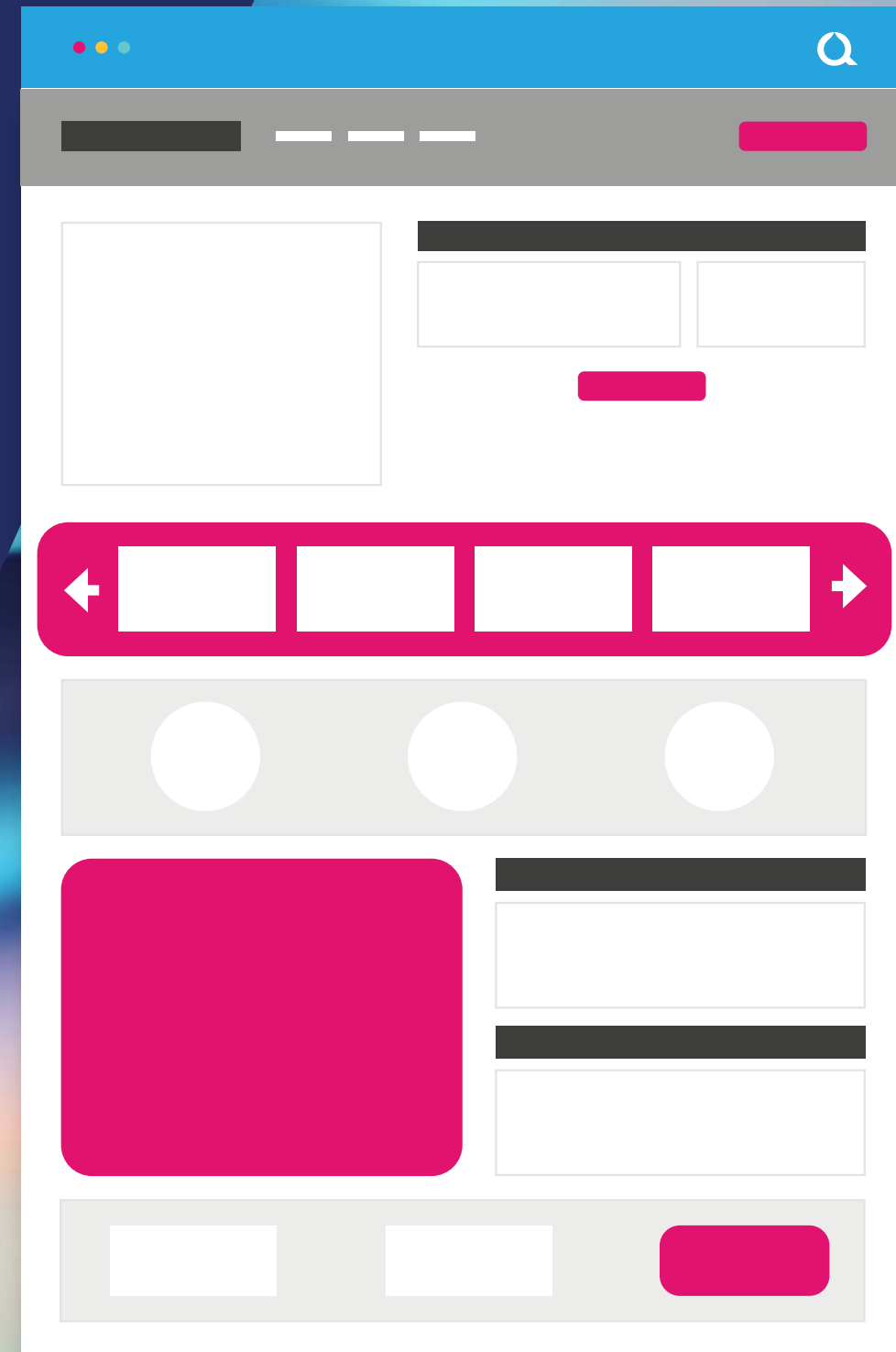
An increasingly popular solution to mitigate the risks of fully decoupling Drupal is a **progressive decoupling strategy**. Unlike a fully decoupled architecture, hybrid implementations insert a JavaScript framework into a Drupal site's front end as decoupled components. JavaScript frameworks continue to consume the REST API; however, certain areas of content can be controlled and rendered by Drupal while others can still take advantage of client-side rendering.

This hybrid approach enables the front-end experience of JavaScript while editorial and technical teams can both continue to take advantage of valuable Drupal capabilities. These decoupled components can be created and implemented as needed in a progressive way. So, parts of the site can be decoupled without needing to commit to either a unified or fully decoupled architecture.

This also means that business users can use the low-code tools to assemble and rearrange the components without needing developer resources.

## ***A PROGRESSIVE DECOUPLING STRATEGY ENABLES YOUR ORGANIZATION TO MANAGE SPECIFIC PAGE ELEMENTS IN DIFFERENT WAYS.***

Here, the decoupled components in pink can be managed per page as needed. The application can selectively choose how much of any particular page to decouple based on the needs of the experience.



## CASE STUDY

# GEORGIA TECHNOLOGY AUTHORITY

**The Georgia Technology Authority is the digital services agency that services the state of Georgia. With Acquia's help, the GTA team devised a strategy to migrate, rebuild, and redesign the state of Georgia's outdated CMS, and transform it into a single cohesive system.**

They built a **multisite architecture** that promotes consistency and flexibility across all agency sites on Acquia Cloud Platform. With this strategy, they were able to migrate content efficiently, while creating a custom responsive, mobile-first search application.

Within 12 months, Georgia.gov launched 55 sites on the Acquia Platform, and officials estimate that the move to Drupal and the Acquia Platform will generate savings of \$4.7 million over five years. This move allowed Georgia.gov to free itself from managing at least 20 servers, and provided a standardized approach for managing all of their web properties more efficiently.

Since site launch, Georgia.gov has continued to partner with Acquia to move government agencies "beyond the browser" by leveraging the hybrid CMS approach.

Over the course of a three-month project, **Acquia built an Alexa skill** that anyone with an Amazon Echo device can take advantage of, whether it's to know everything about food stamps in Georgia or to perform simple inquiries like transferring an out-of-state license, acquiring an early election ballot, or registering for a fishing license.

The content for these questions is provided by the Georgia websites via the API. This information can be updated in a single place, and everywhere that uses that data is instantly updated as well.





***SUMMARY AND TAKEAWAYS***

***DRUPAL LETS YOU  
CHOOSE THE RIGHT  
APPROACH FOR  
EACH PROJECT***

In an environment where brands must deliver a streamlined, optimized, and rewarding digital experience for their audience through whichever “glass” users choose to interact with content, it’s important to leverage a CMS approach that works well for both marketers and web developers.

Using a headless Drupal architecture, where the back end is decoupled from the presentation layer, can certainly give web development teams the control and flexibility they need to deliver creative solutions for audiences across many channels and devices.

But if, as many organizations are finding, this approach either increases the number of technical issues or restricts marketers from implementing content and campaigns the way they want to, the headless CMS route may not be ideal.

Thankfully, there is another approach growing increasingly popular among both kinds of teams.

Hybrid headless Drupal enables you to combine the benefits of a unified system with the advantages of a headless CMS. Getting that balance right will make it easier to build, manage, and optimize your content and data so you can deliver exceptional digital experiences through a variety of devices. Even if you don’t get the balance quite right at first, a hybrid CMS allows you to calibrate your approach until it’s optimized.

Adopting hybrid headless Drupal enables web development teams to:

- ✓ **Work in an independent, yet collaborative, manner**
- ✓ **Streamline content management**
- ✓ **Expedite production timelines**
- ✓ **Deliver vital information through a growing number of different interfaces**
- ✓ **Adapt to future technology quickly**
- ✓ **Optimize digital experiences**

For development teams looking to deploy Drupal as a headless CMS, Acquia provides a full platform of tools and capabilities to support every stage of the developer and marketer workflow. Founded by the same open source pioneer who founded Drupal, Acquia is an open digital experience company that enables ambitious brands to embrace innovation and create customer moments that matter.

**WANT TO LEARN  
MORE ABOUT HOW  
A HEADLESS OR  
HYBRID DRUPAL  
CMS CAN DELIVER  
GREATER FLEXIBILITY  
FOR YOUR TEAMS?**

**REQUEST A DEMO**



# Acquia

ACQUIA.COM

## ***ABOUT ACQUIA***

Acquia empowers the world's most ambitious brands to create digital customer experiences that matter. With open source Drupal at its core, the Acquia Digital Experience Platform (DXP) enables marketers, developers and IT operations teams at thousands of global organizations to rapidly compose and deploy digital products and services that engage customers, enhance conversions and help businesses stand out.

