Acquia
EXPERIENCE DIGITAL FREEDOM

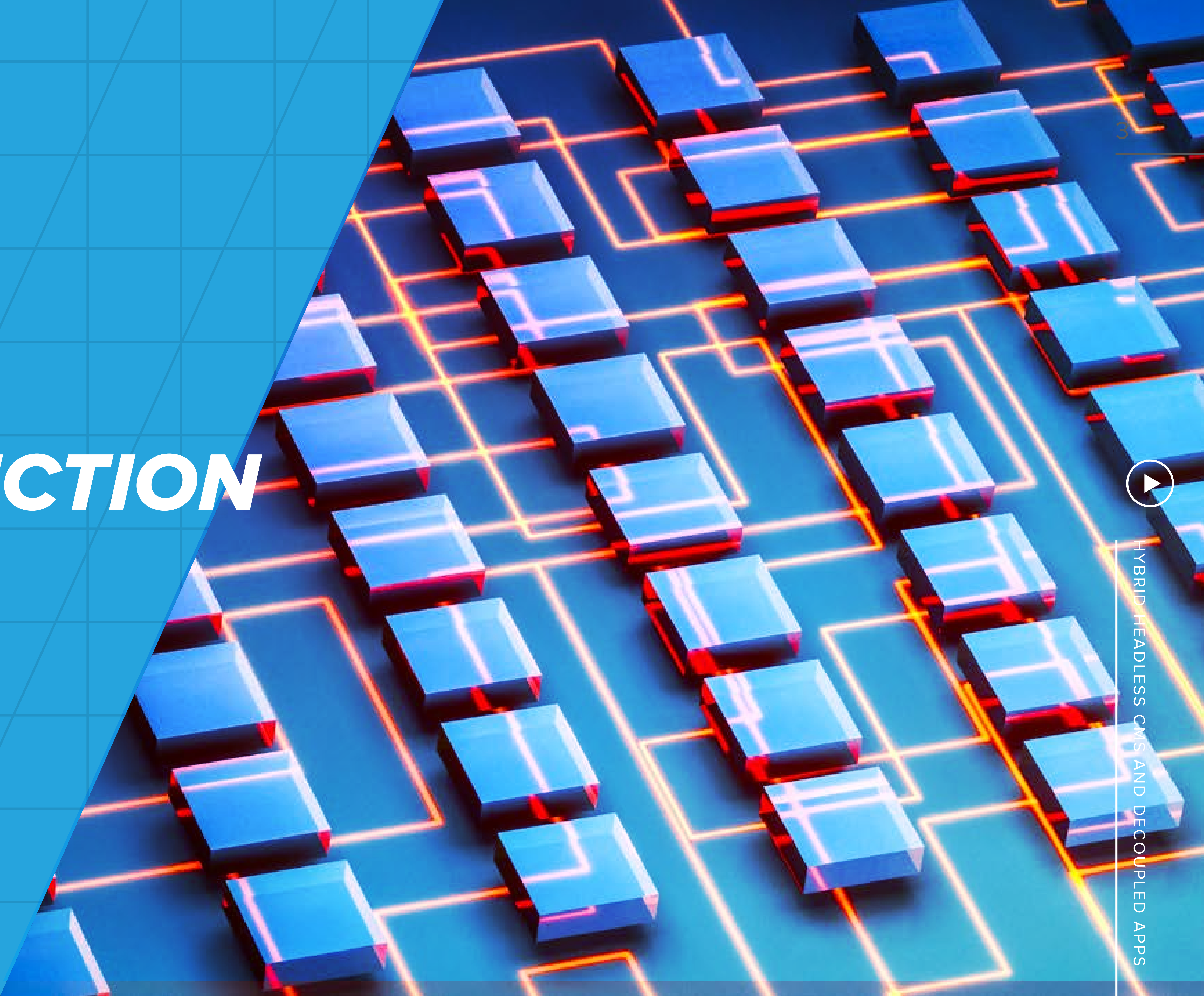
A HOW-TO GUIDE TO HYBRID HEADLESS CMS AND DECOUPLED APPS

Choosing the right architecture to support web-based applications

TABLE OF CONTENTS

SECTION 01

INTRODUCTION

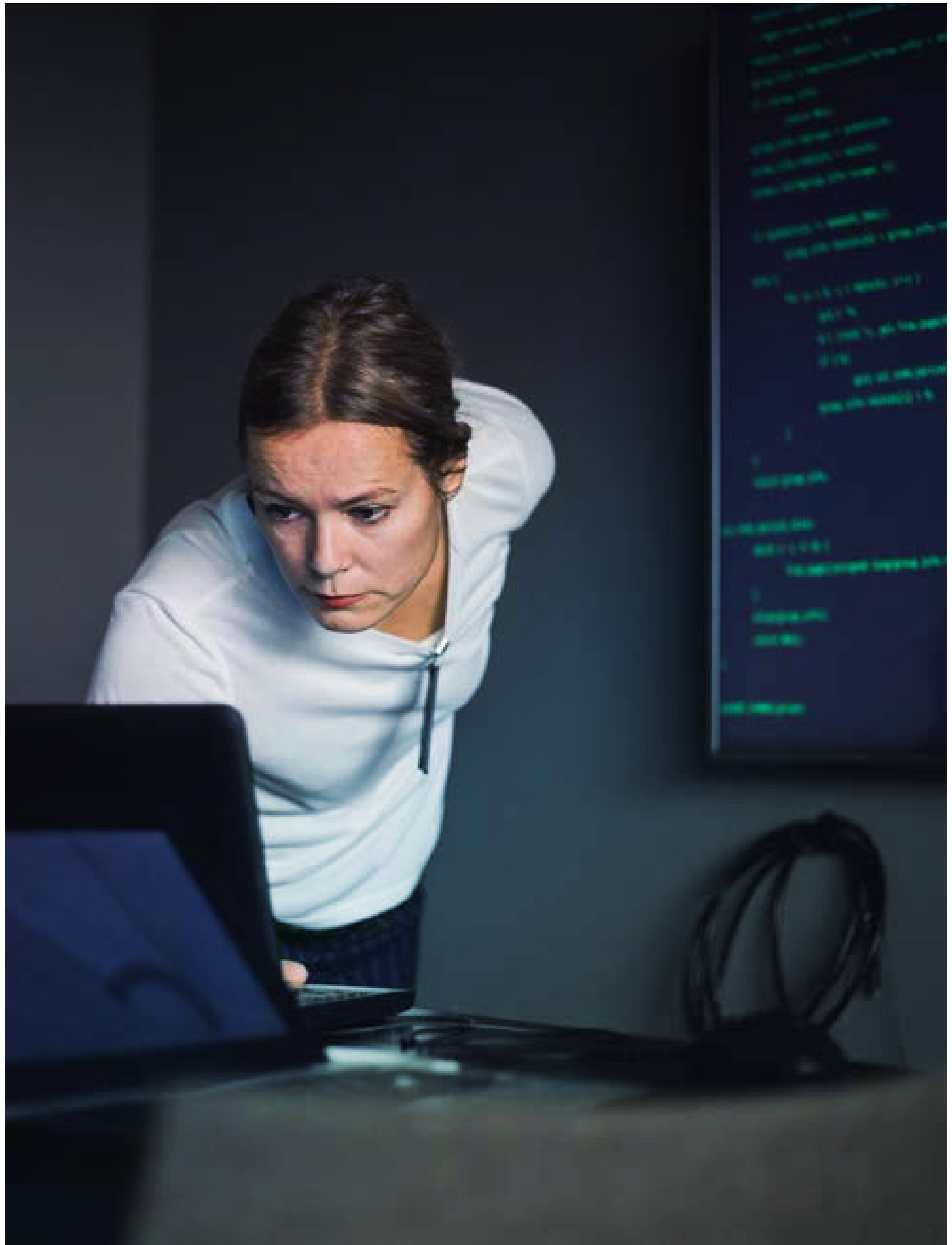


For most of the 21st century, most websites have been built with a unified architecture. In this model, a content management system (CMS) manages both the front end and the back end of a web application. This has proven resilient for many reasons: caching, security, web application advancements and more.

Yet, the primary reason is because of the need to provide low-code tools and web-based user interfaces (UIs) for non-developers. These tools allow them to control the experiences they are responsible for. In fact, the rise of low-code tools continues to grow as more people need the ability to manage web experiences. At the same time, the rise in new interfaces and application types has generated a dizzying array of channels to support. These digital channels provide new challenges that often require high-code solutions.

Often, the ideal solution is to break the unified architecture into one or more "decoupled" pieces. This typically relies on the CMS to manage the back-end data and allows another application to manage the front-end experience. In other words, this is a headless CMS back end paired with a decoupled front end.

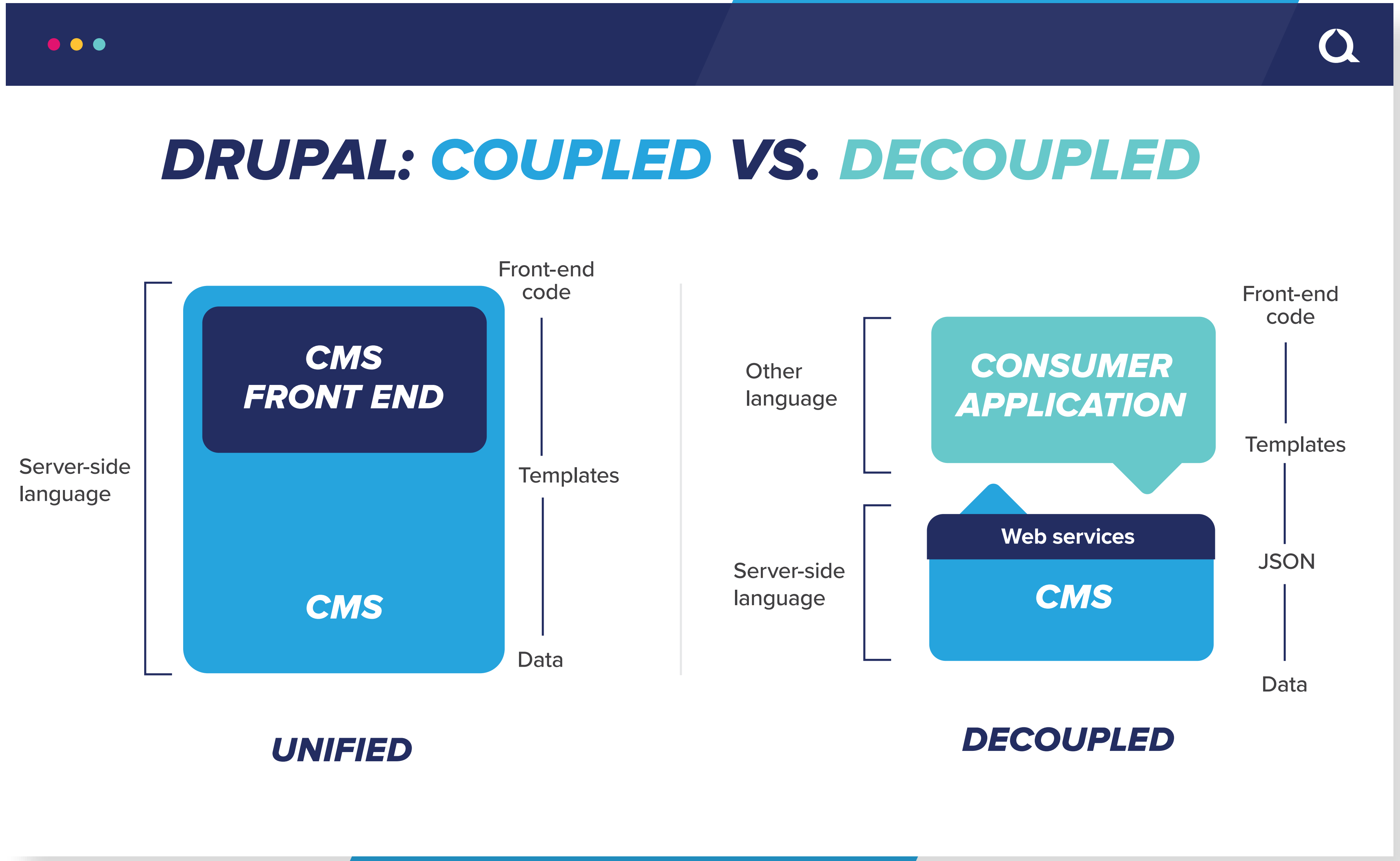
Forrester and Gartner highlighted headless CMS and decoupled architecture as an important capability back in early 2016. Recently, Gartner has modified their advice to focus on a hybrid CMS that is API-first, and not API-only. This shows the increasing importance of developing digital applications that share a common platform as we move further into the 21st century.



THE DECOUPLED EVOLUTION

Websites in the past were built from monolithic architectures that deliver content through a templating solution tightly coupled with the CMS on the back end. Agile organizations crave flexibility and strive to manage structured content across different presentation layers. Accomplishing this efficiently requires that teams have flexibility in the front-end frameworks that dominate the modern digital landscape.

That's why decoupled and headless CMS have taken off. That's why you're here. But now you need the right technology to support the next phase of the web and beyond.

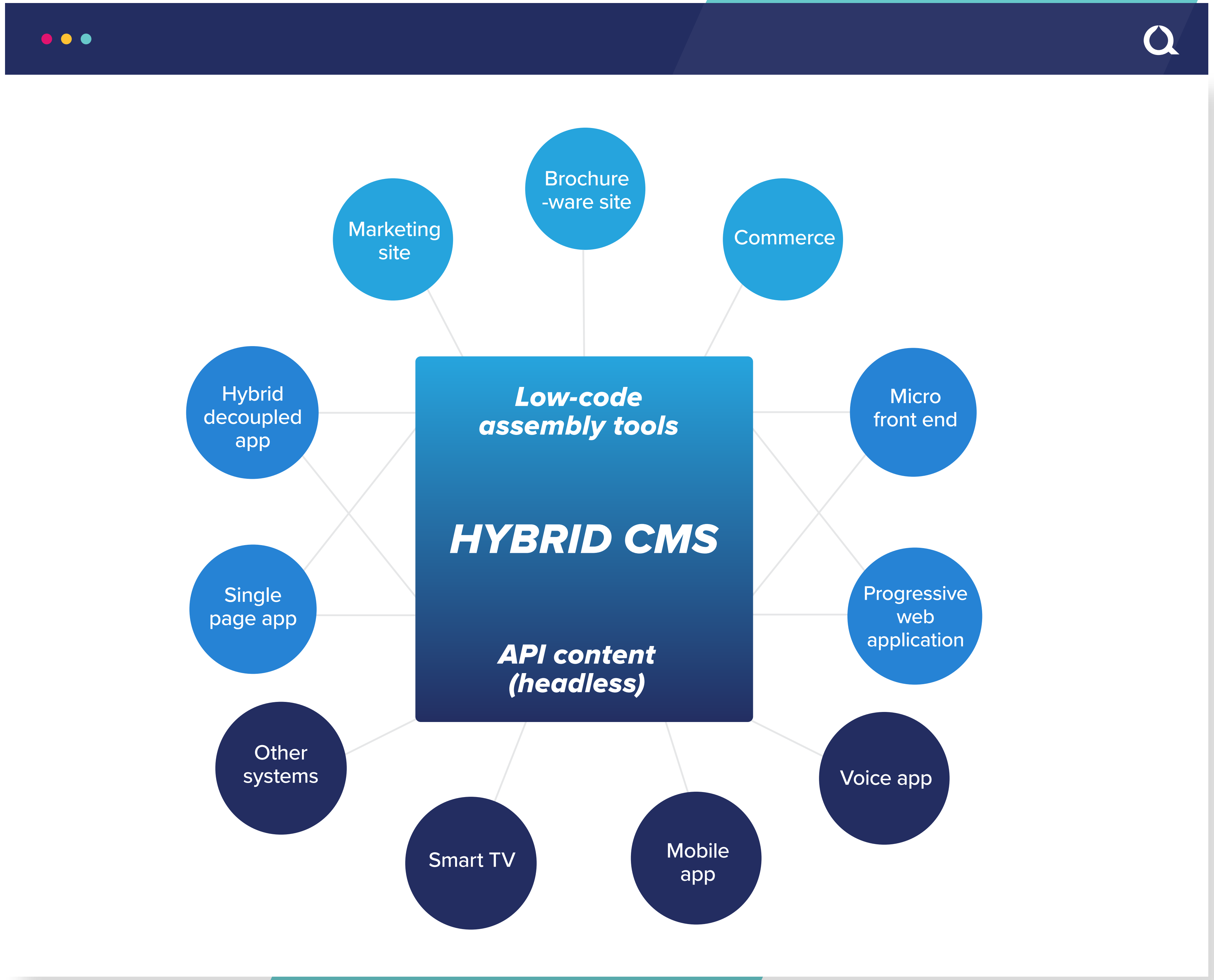


SECTION 02

DEFINING HYBRID CMS

So first thing's first: what is the difference between a traditional CMS, headless CMS and a hybrid CMS?

- **Traditional CMS:** Users create content through an editor and store it in a database (the back end). That content is then served into a front-end rendering layer that is tightly coupled to the back end.
- **Headless CMS**: Users create content through an editor and store it in a standalone database fronted by APIs. The content is retrieved by an entirely separate front-end rendering layer via those APIs.
- **Hybrid CMS:** Blends the traditional and headless CMS. Users create content through an editor and store it in a database. The content can be served flexibly either through the existing front-end rendering layer or retrieved by an entirely separate front-end rendering layer via APIs.



In all cases, front-end templates define how content is displayed in the digital application. What is changing is the landscape of languages and frameworks available to the task. In order to meet business (and consumer) demand for a consistent experience across channels, teams need to have expertise across multiple front-end frameworks, and in the last several years that list has changed.

Popular languages for traditional website development include PHP, .NET and Java. However, the popularity of Node.js as an application server has created an explosion of activity. According to the 2020 Stack Overflow survey, JavaScript is still the dominant programming language for front-end web development, and has been for years.

When considering a traditional CMS architecture versus headless or decoupled architecture, it is important to understand the use cases best suited to the architecture. For example, the majority of projects that use JavaScript front-end frameworks on Node.js web servers typically lean on the inherent strength of Node.js: real-time or asynchronous functionality. Use cases usually revolve around the ability to maintain single threading in a non-blocking way. A common example would be a React app, which would use a combination of multiple views or API endpoints without the need for a page refresh.

Other common examples of this functionality would include:

- **Dynamic API Factory**
- **Real-time or data streaming**
- **Chatbots/chat clients**
- **Messaging over WebSockets**

If your organization is evaluating a decoupled architecture and you have any of the following requirements, a decoupled architecture may **not** be a great fit:

- **Full-fledged editorial experience for your content team**
- **Frequent needs to manipulate display and layout**
- **End-to-end preview of the application prior to launch**
- **Minimal developer resources needed to maintain your digital application infrastructure**

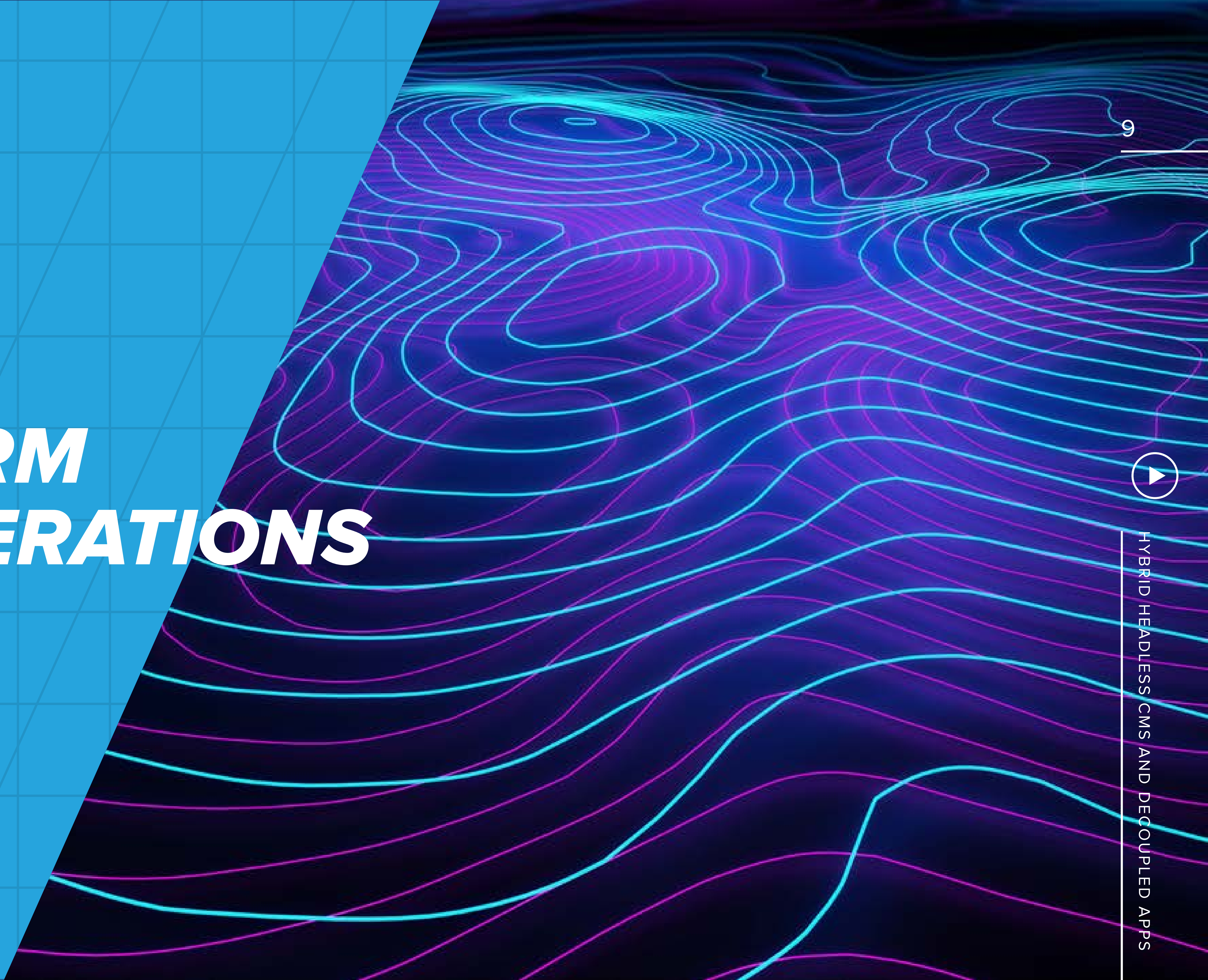
This is why the default recommendation is to use a hybrid CMS. A hybrid CMS is API-first, and can be used for both headless and traditional implementations. This means that a single toolset can be used and reused for different projects internally without needing to evaluate, purchase and retrain on different solutions.

Drupal is an open source, API-first CMS and one of the most powerful hybrid CMS solutions. API-first is what really enables decoupling to happen because the application "phones home" to the CMS to retrieve content. Content can sit in one place and then be distributed outward. Instead of having a siloed system for a website, you have a hub and spoke model. The hybrid CMS is the hub and the spokes are single page apps, smart TVs and even other back ends for other apps.

This also means that Drupal will allow you start with traditional and "add on" headless capabilities, or vice versa. The flexibility gives your team choices without sacrificing security, performance or development speed.

SECTION 03

PLATFORM CONSIDERATIONS



Digital platform teams require a system that will help them support web, mobile web, mobile apps, chatbots, voice-activated applications and more. Options to support these channels are split among the technology approaches above. Traditional CMSes, like Adobe, Sitecore, Episerver or Joomla, are best suited to monolithic architecture. There are a number of newer, API-only (headless) CMS systems gaining traction like Built.io, Contentful and Prismic.io. And finally, there are API-first options, like Drupal, that are able to deliver on the promise of decoupled architectures. The challenge is determining the right option for not only your current use cases, but for your organization's strategy roadmap.

The decision is driven by the changing digital landscape. Chatbots, voice-enabled apps and augmented reality are poised to become very popular. The question is how this will impact the way your organization addresses your participation in the growing digital landscape.

Will you choose to build everything from scratch and carry not only the maintenance burden but also a radically slower velocity? Or will you choose off-the-shelf point solutions that are not exactly what you need and you will struggle to evolve with them? The ideal platform is one that allows you to compose your specific solutions from a library of reusable components and services, while still being able to adjust and customize as needed. This balance of speed, governance and flexibility is the key to success in the modern digital space.

By choosing a reliable hybrid CMS, you can not only support multiple use cases with a single toolset, but you can also allow teams to work on related projects without getting in each other's way. The digital marketing team can use the low-code tools to build and modify the main website while your mobile application team can tap into the same content via the API. Hybrid enables you to build differently, so that the silos of web and mobile are broken down and you can support a unified digital apps team.

Once this is accomplished, the team can organize around content structure versus presentation. This can quickly result in freeing front-end developers from back-end obstacles.

With a common toolset in place and bottlenecks addressed, teams can now employ continuous development methodology. Continuous development automates the build, integration, test and deployment stages of application development. This allows development teams to deliver new applications and features to users faster and more efficiently. In a hybrid website or application development, this approach gives both front-end and back-end teams the independence to develop structured content models and modern presentations that best meet the objectives of the project.

In order to reap the advantages of the architecture, you need a platform that is designed to support these use cases. With continuous development, teams benefit from a common development toolset to aid in communication. Support for application hosting for the front end and back end on a single platform will aid you through the consolidation of infrastructure suppliers and a single support structure and SLA.

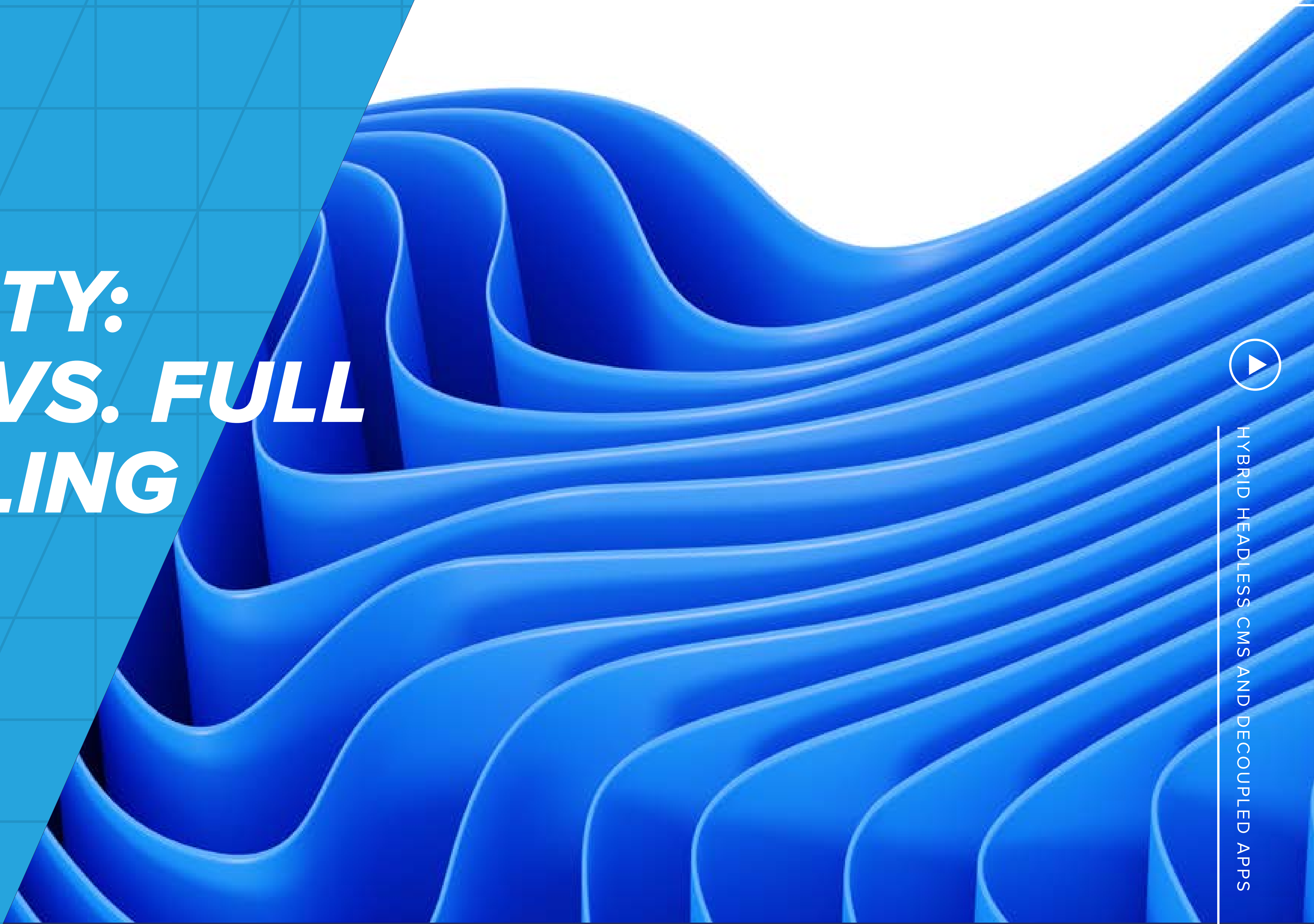
In order to deliver a complete omnichannel digital experience, your platform must either easily integrate with or include:

- **Front-end framework (e.g., Node.js)**
- **CMS (e.g., Drupal)**
- **Cloud-based development tools**
- **Personalized content delivery**
- **Customer journey orchestration**
- **System integration (e-commerce, marketing automation)**
- **A robust API-first approach**
- **Support for application and infrastructure from a world class team**

The benefits of creating decoupled applications with JavaScript on a single platform are common development tools and a common UI for your teams to use. If you're using some sort of build and test automation, they can both be using the same tools. Teams are able to work more efficiently by sharing unified management consoles and developer tools.

SECTION 04

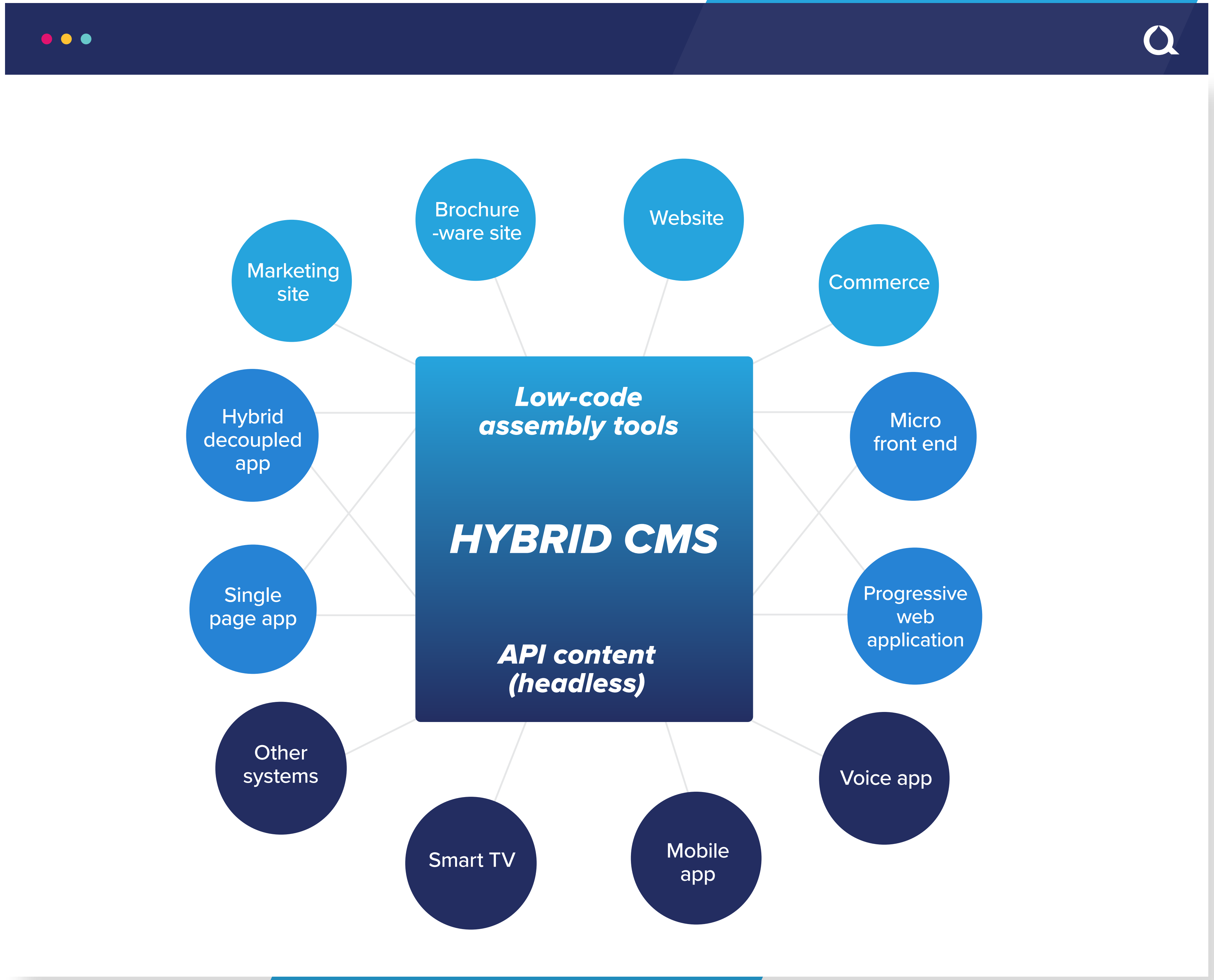
FLEXIBILITY: PARTIAL VS. FULL DECOUPLING

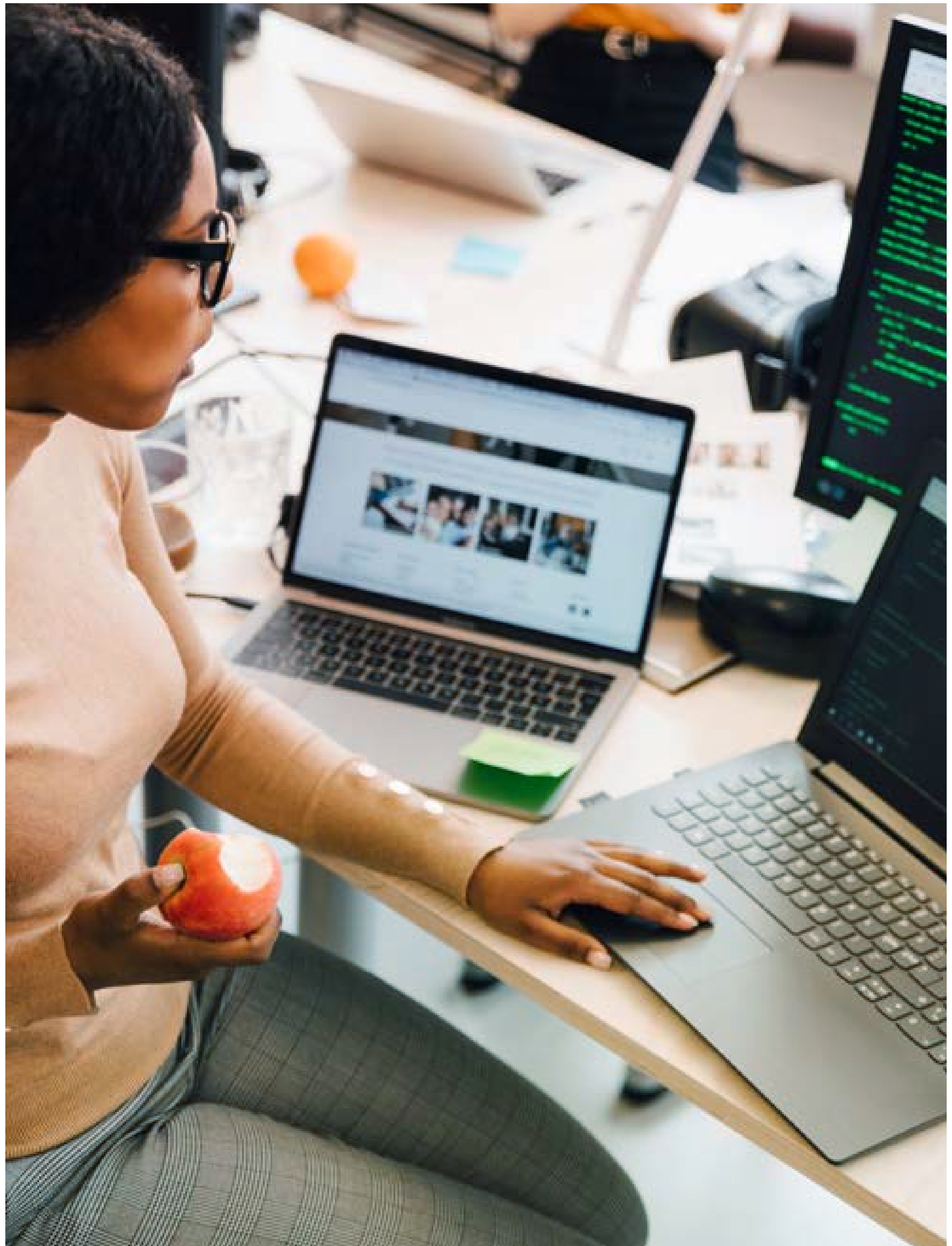


For some projects, you choose a full-stack, or unified, Drupal application as the primary solution. In a traditional model, the entire front end is managed by the back-end server. However, there is growing interest in more interactive technologies for your site or the additional digital channels available. A partial decoupling approach can use JavaScript components to display specific interactive components in the browser, but maintains cacheable content and additional workflows within the Drupal back end.

Partial decoupling mitigates many of the limitations of a fully decoupled architecture by leveraging low-code tools for managing the presentation layer and inserting decoupled components where they are needed. This allows the team to compose experiences from a library of reusable components and deliver results without needing to deploy code. Low-code assembly empowers the non-developer to safely and reliably manage experiences using self-serve tools in the back end.

Full decoupling puts more power into the hands of the developer and, if deployed properly across a broader team, can lead to a more effective and efficient development workflow. The CMS is put into a fully headless mode and is only used for creating and managing content and data for the decoupled front-end application to consume.





PARTIAL DECOUPLING

Partial decoupling combines the benefits of traditional Drupal applications with the benefits of front-end frameworks using decoupled JavaScript components that are assembled with Drupal in combination with standard CMS content. Using the Component module, JavaScript developers can easily add their components without needing to know any PHP or Drupal specifics. They simply need to create a formatted YAML file with the component data and Drupal will auto-discover and load the component with all of its dependencies.

Once it is discovered by the CMS, the content creator can easily use the decoupled component like any other piece of content, and the visual page builder will let them compose or assemble the content with a simple drag-and-drop interface. This is a great way to decouple parts of the experience as opposed to all of it.

Partial decoupling enables teams to:

- **Rely on Drupal to render and deliver the Skeleton boilerplate and dynamic sections of the page**
- **Decouple the components of the page that need to be dynamic or uncached**
- **Leverage the BigPipe Drupal module to speed up page loads while decoupled and dynamic components populate asynchronously**
- **Offload some front-end rendering to reduce server resource usage to allow pages to load faster**
- **Optimize API requests to arrive at smaller, lightweight requests**
- **Use and contribute to a shared library of JavaScript components**

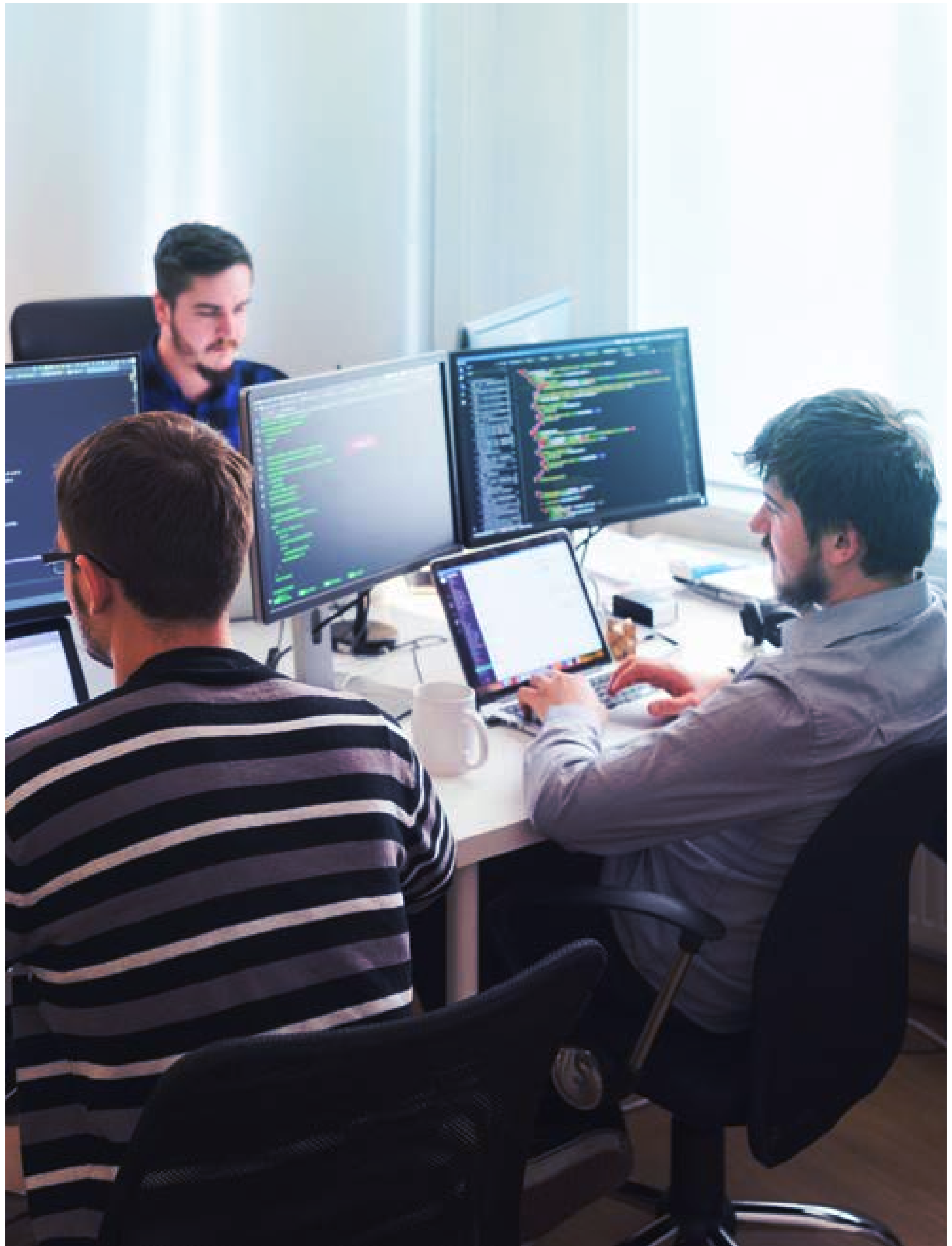
FULL DECOUPLING

Full decoupling is what we think of when we envision a React or other JavaScript application. This is a way of moving much of the experience management responsibility to the developer team. In some cases, this can make it easier to manage multichannel communications and in other cases, it can be the only viable way to view the content (like on a smart device).

In order to balance the infinite level of variation across completely custom applications, it is important to outline the toolset that will be used for building and managing all decoupled applications. Consideration needs to be given to the possibility of needing to support one or more native applications along with web applications.

Most major JavaScript frameworks will work for the average use case, but you should also look at what your developers are comfortable with and how the different tools are evolving and growing. Ideally, you should be opting for multipurpose tools and standard architectures that are as simple as possible. You want to remove complexity where you can and provide a standardized method for the entire team to use for application development.

Decoupled apps bring together the best of the old web (pre-smartphone) and the best of what the current web has to offer now and in the future. But all these shiny new things need to be developed in a way that lets them connect. Looking at how to build decoupled applications, one thing has become clear: you need JavaScript. JavaScript and its frameworks, like Angular, React and Vue, have been ranked by StackOverflow as the most popular scripting language since 2013 in a survey of over 50,000 developers.



SECTION 05

CHOOSING THE RIGHT JAVASCRIPT FRAMEWORK

There are many JavaScript frameworks to choose from. There's React, one of the most popular frameworks that's been added to Drupal core. There's Express.js. There's Vue. There's Svelte. There's Angular. The challenge with JavaScript from a developer's standpoint is it's kind of a wild, wild west. The choice of framework rests on use case and developer preference.

However, if using Drupal, then it makes sense to look at the existing frameworks that have supported integration patterns and strong sample code. For example, the Contenta distribution has several sample decoupled applications to work with, and the Next.js integration offers an optimized integration that provides a real-time preview of the decoupled application.

Some resources to consider:

- **Acquia CMS**
- **Drupal State JavaScript SDK**
- **Drupal Next.js for React**
- **Drupal Tome - static site generator**
- **Drupal Gatsby**
- **Component module**
- **Decoupled Pages module**
- **Decoupled Kit module**
- **All decoupled-related modules**

For teams that have made the choice to go with the decoupled architecture for a specific solution, the typical question is what to do about the decoupled application. When they're running the front end on a completely different stack, in this case a JavaScript stack that includes Node.js as the runtime, those people have to set up their stack that supports the CMS. Then, they go somewhere else, like Google Cloud or Amazon Web Services, and they set up this front-end runtime to run over there.

When Acquia launched our Node.js support back in September of 2017, we took a lot of the complexity out for our customers because now they've got one UI to run both the front- and back-end stack. We designed our support for Node.js around delivering the optimal stack configuration for developing decoupled applications. Rather than building a complete Node.js stack and a LAMP stack, decoupled applications on Acquia Cloud Platform are supported by a full LAMP stack and the Node.js runtime all on a single platform.

This combination of support for a truly hybrid CMS along with decoupled application support makes the Acquia platform a powerful tool that you can use to create traditional, headless, decoupled, static and low-code applications, all on the same platform.

SECTION 06

BEST PRACTICES WITH DECOUPLED API: ACQUIA ENGAGE

In October 2017, just as Acquia was releasing support for Node.js on Acquia Cloud, we acquired a new customer: ourselves.

Acquia was facing the same situation that many of our customers face on a regular basis: a requirement for a digital experience tied to an event. In Acquia's case, it was building a decoupled application for our annual conference, Acquia Engage.

The first step in a successful decoupled project is aligning on requirements. In the case of Acquia Engage, they were as follows:

- **Provide real-time updates with presentations and speaker information**
- **Showcase information about Engage Awards finalists by category**
- **Showcase other types of content related to the conference**
- **Integrate content from standard APIs, such as JSON, into the application**
- **Work responsively on multiple devices and screen sizes**

Once all the requirements were set, the next step was to understand the decoupled Drupal workflow. To do this effectively, Acquia:

- **Used development to help contrast assumptions around level of effort based on tasks**
- **Standardized workflow along with deployment requirements when building two applications that work in parity**
- **Streamlined communications around potential issues and understood how to troubleshoot issues with the build based on the technical level of the audience**

The Acquia Engage application was built in JavaScript to execute on a Node.js runtime environment in Acquia Cloud. It was important to test any limitations or technical opinions of Acquia Cloud's Node.js hosting before getting started.

This included documenting implications of how a Node platform built for decoupled Drupal is different from a standalone Node hosting offering. Acquia also had to adjust the application based on requirements of automating a deployment cycle based on Acquia pipelines.

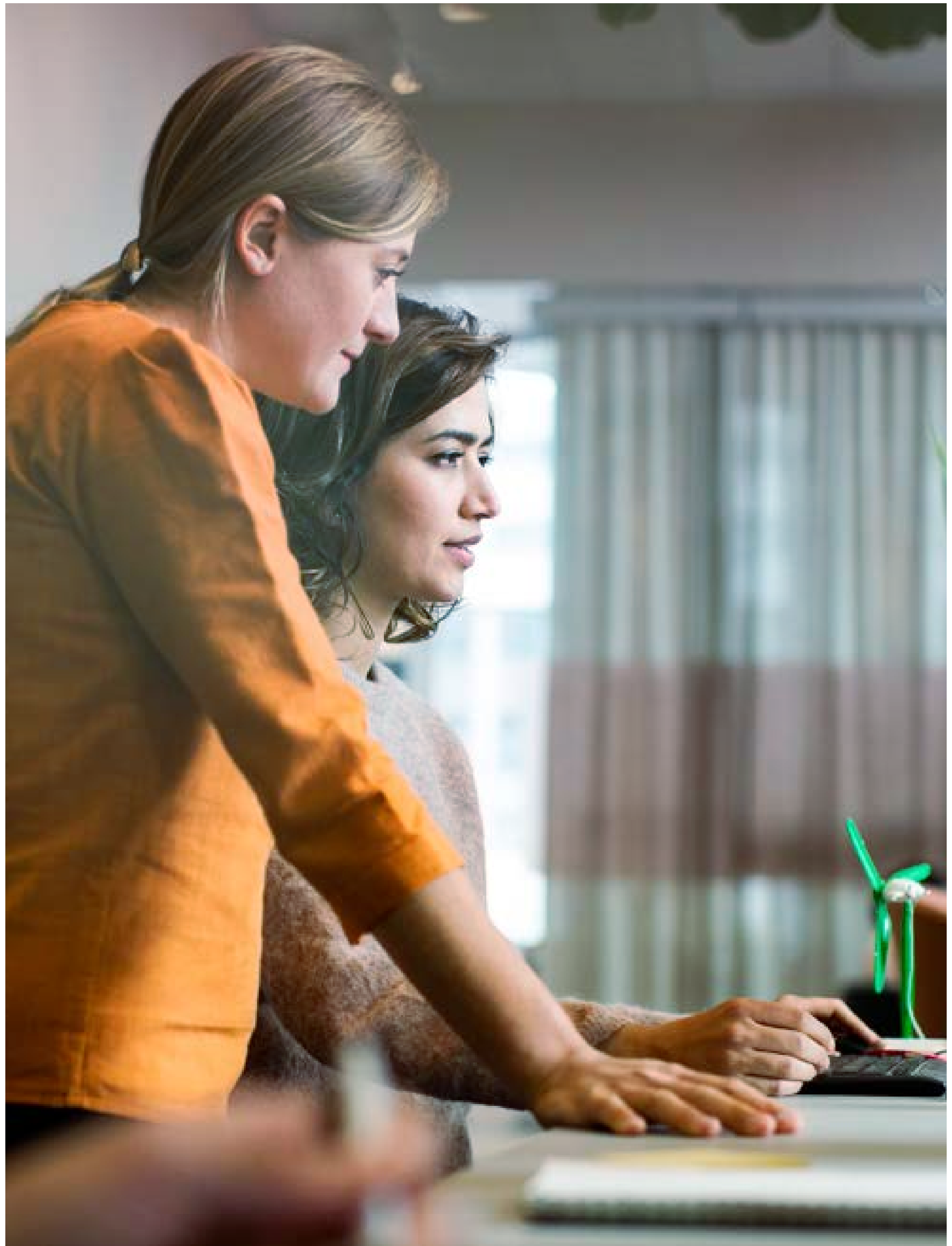
Once limitations and technical options were established, a content workflow was enabled for the non-technical content authors who would be working with and updating the content within the Acquia Engage app. To maximize efficiency, the marketing team needed the ability to manage the content with a standard CMS-managed workflow and provide real-time updates without the need for code deployment. In addition, the marketing team needed authorization to upload media assets based on type of content. To meet these requirements, Acquia needed to allow for customizations integrated with Drupal that carried over to the JavaScript application.

Results:

- **Successfully supported ~250 users per day during conference**
- **Cut development time by 30%**
- **Node.js processes allowed for real-time data to be presented to Acquia Engage attendees through the Engage App**
- **Editor workflow increased by 35% during conference updates**

SECTION 07

CONCLUSION



Decoupled architectures are foundational to the direction digital teams are taking applications into the 21st century. In fact, Forrester stated: “the microservices approach is the future of digital experience architectures.” Despite that, moving to this pattern requires thoughtful decision.

Understand your use cases or digital applications and make sure you are aligning to the best architecture to meet your needs. If your priorities include a heavy reliance on delivery of content over API, real-time data or omnichannel support, decoupled Drupal should be seriously considered. If your objective is a blazing fast experience for the web or other digital application, leveraging Drupal as a service to a Node.js runtime makes it easy to deliver an amazing front-end experience.

Despite the seemingly endless benefits, ensure you weigh the tradeoffs of managing a decoupled stack. Ensure your partners and/or internal teams can deliver the support needed to keep your applications running optimally. Make sure your team is ready to build applications differently. Decoupled can introduce new development tools, languages and approaches. Make sure your teams are prepared for the coming changes. Lastly, make sure you evaluate your platform approach based on the support of both your front-end and back-end teams.

TAKE THE NEXT STEP

Now that you've learned how to choose the right architecture for your current and future applications, find out how Acquia CMS can support whichever approach you take.

LEARN MORE ▸

ACQUIA.COM

ABOUT ACQUIA

Acquia is the open digital experience platform that enables organizations to build, host, analyze and communicate with their customers at scale through websites and digital applications. As the trusted open source leader, we use adaptive intelligence to produce better business outcomes for CX leaders.