How to power Brand Experiences using Single Page Applications in Adobe Experience Manager
Executive summary

As businesses evolve to meet consumer expectations, commerce platforms are evolving to meet the needs of marketers. Today, one of the biggest challenges for marketers is to find scalable, mature technology solutions to architect and curate experiences at speed and as a consumer would use them.

The way a page loads from the server has created a fundamental shift in the building of digital experiences. Rather than each user action requiring an entirely new page to be loaded from a server – an inefficient, jarring user experience – marketers today are able to provide users with just the information they need to complete a transaction.

It has paved the way for more complex and dynamic digital experiences. The creation of Single Page Applications, or SPAs, as an alternative to the more traditional “multi-page applications,” has boomed as a result of its ability provide streamlined and fast experiences for the consumer, whilst also improving the experience for the site author. The growth in SPAs is also mirrored by the growth in the Headless Commerce model, with increased personalisation providing huge consumer and brand benefits.

Adobe’s product suite offers several products that enable brands to offer superior user-centric experiences by creating ecosystems where businesses can connect and track the entire user journey. For the purposes of this paper, while these concepts apply to multiple products available on the market, we’ll look at them through the lens of the Adobe Marketing Cloud ecosystem.

Read this white paper for a detailed overview of what Single Page Applications are, how they can be managed as part of a Headless Commerce experience, and how SPAs can be implemented using Adobe Experience Manager.

You will find an introduction to SPAs and how they connect with Headless Commerce models. You will also find the information you need to determine whether you need to act now and set up SPA architecture to deliver on consumer expectations and achieve growth.

You also have a Start up for Success kit for your technical architects with 7 key technical considerations that you need to make when integrating an SPA into a Headless Commerce experience, but also the factors that are involved.

Get your head around the terms and acronyms in this report by reading our glossary.
Contents

SPAs: in the know

What CMOs & CTOs need to know

• What are SPAs and how do they connect with Headless Commerce models?
• Should you build a SPA or create a traditional marketing site?
• How can AEM support the build of an SPA?
• When & how proprietary solutions can enhance AEM and to solve your unique needs

Start Up for Success

What technical architectural considerations should you make when integrating SPA into a Headless Commerce experience?

• Decide On a JavaScript Framework
• Choose an asset compilation strategy; a custom solution or AEM’s out-of-the-box client libraries
• Determine how to plug in widgets or a SPA into your application
• Understand how to leverage AEM to provide data to a SPA experience
• Leverage AEM 6.4 SP2 to create and manage SPA content via the authoring experience
• Enhance SPA Architecture to meet SEO Requirements
• Determine the configurations to a SPA using custom OSGi settings

What other factors should be considered when defining SPA architecture?

• Using an API Tier in front of a SPA
• Serving personalized content by using Adobe Target as part of your SPA
• Allocating financial budgets judiciously to utilize the expansive Adobe ecosystem
• Playing to your team’s strengths to maintain development velocity
• Integrating SPA code as part of the code ecosystem
What are SPAs and how do they connect with Headless Commerce models?

In the context of websites, a single page application, or SPA, takes advantage of modern browser APIs that enable a website to behave more like a native mobile or desktop application. SPAs do this by serving a templated shell of a site to the user on the initial load, along with any other assets that will be required based on subsequent user actions. Any data, for example user data or product data, that is required as a result of those user actions is retrieved and displayed to the user separately and as necessary. It enables an uninterrupted user experience - no load waiting times, more dynamic content and increased opportunities for personalisation through Headless Commerce integrations. In addition to providing the user with a superior experience, this enables data being returned to the user to be structured so it does not need to be re-worked to suit the interface it is being served through. This means that the data that you use to build and manage an SPA can be used for other brand experiences - not just websites.

SPAs align with the Headless Commerce model[1] where the user is at the centre of multiple touchpoints; web, mobile applications, wearables, virtual assistants, etc. Using the Headless Model, a marketer can manage elements of their business - product catalogues, promotions, user data - in tools designed for its own business processes while designing and deploying user experiences unique to those touchpoints.

Figure 1: An illustration of the Headless Commerce model
Traditional ecosystem models create brand experiences in silo, so consumer data is often disconnected. But the Headless Commerce model is user centric and enables brands to create personalised experiences by feeding data back to a single platform. Adobe’s product suite offers several products that enable brands to offer superior user-centric experiences by creating ecosystems where businesses can connect and track the entire user journey.

One of the challenges for marketers is to find scalable, mature technology solutions to architect and curate these experiences at speed and as a user would use them. For the purposes of this paper, while these concepts apply to multiple products available on the market, we’ll look at them through the lens of the Adobe Marketing Cloud ecosystem.

Should you build a SPA or create a traditional marketing site?

Before we dig into the key areas to consider, there remains several things to think through before deciding a SPA is the right type of application for your business needs.

An SPA will not be needed if:

- The application will largely be comprised of static content.
- There will be little to no interactive features.
- Your team of developers have limited React or Angular experience.
- There are stringent SEO requirements, and you can’t support a NodeJS server in your application architecture.

If any, and certainly if all, of those conditions are met, it likely is not a good idea to explore a SPA implementation for your business needs. Instead, a traditional AEM implementation that utilizes HTL templates and multiple HTML pages should still provide the tools you need to deliver a robust, authorable site to customers.

How can AEM support an SPA?

When building a website, Adobe Experience Manager (AEM) provides many of these tools out of the box. To create an engaging SPA, AEM can also help with solutions to:

- Visualize SPA content in the AEM authoring experience.
- Address SEO concerns related to SPAs.
• Provide personalized content to a user.
• Consolidate data from multiple APIs, mitigate risk by not relying solely on AEM as an API tier.
• Allocate financial budgets judiciously to utilize expansive Adobe Marketing Cloud ecosystem.
• Play to team strengths and skill sets while maintaining development velocity – a gauge of how much work is completed in a set period of time.

When & how proprietary solutions can enhance AEM & to solve your unique needs

AEM does many things well. Its out-of-the-box tools enable businesses to segment groups with various administrative and authoring rights and it manages multiple image renditions for use across device viewports. On a larger scale, using tools like Multi Site Manager, it allows businesses to create a global CMS experience that can be managed in a central location, with copies made for various localizations, with shared customizable components available across various brands within an organization. Most importantly, it enables website authors see the site as a user would before publishing it using industry best drag and drop user interfaces.

However, there are some situations where it is prudent to augment AEM with custom solutions, such as when dealing with external API tiers, session management, and SPAs.

In this section, we’ll highlight where and why Isobar sees value in utilizing custom solutions (for example external API tiers, session management and SPAs), and how that approach will evolve as the AEM product continues to evolve as well.

What technical architectural considerations should you make when integrating SPA into an AEM experience?

By design, an SPA behaves and feels more like a native application. It does this by making only one page load request from a server, and then making subsequent requests only for data as its required. In contrast, AEM has traditionally utilized a page refresh as a user navigates throughout a site to see updated content. While recent versions of AEM 6.4 provide new ways to integrate SPAs into the site authoring and end-user experience, brand technologists need to consider the Javascript frameworks, asset compilation systems, widget integrations, and the impact on SEO and SPAs, to end up with an integrated authoring experience.
An integrated experience enables the author to be able to see content updates to the SPA in authoring mode instead of just in publish mode (a more headless context). These considerations are explained below.

Figure 2: SPA content and data requests when integrated into an AEM application.

AEM + SPA: The presentation layer makes an initial request to retrieve all HTML/CSS/JS needed for the application shell from AEM. Subsequent requests to AEM retrieve additional content in JSON format.
1. Decide On a JavaScript Framework

To create a robust, full-featured Single Page Application, solution architects must use a JavaScript framework that defines the entire application design. While it’s possible to create a custom framework by piecing together smaller libraries, there are several mature, open-source frameworks that can be used to create an SPA. In all but very few cases, it makes sense to choose from one of those frameworks; i.e. React, Angular, Ember, etc.

For the purposes of this paper, we are using React and its ecosystem of libraries to facilitate SPA integration into AEM, but the concepts can be applied to any framework.

The React developer community has exploded over the last several years. With support from Facebook, documentation is robust, while feature upgrades and bug fixes are continuously integrated into the library. Combined with Redux to manage data via a single state tree, the modular, component-based React library lends itself extremely well to responsive, component-based designs. In addition, React has been at the forefront of enabling server-side rendered HTML; making SPAs search-engine friendly and ensuring a user can see content as soon as the first web page is loaded from the server.

2. Choose an asset compilation strategy; a custom solution or AEM’s out-of-the-box client libraries

As user experiences have shifted the burden of business logic to the browser, JavaScript codebases have grown exponentially. That complexity is managed by ensuring the code is written in a module context specific to the component that the code relates to.

AEM provides a means to do this via the “clientlibs” approach, that enables the site author to manage all the Javascript and CSS resources within an application. That approach can be useful when developing simple interfaces, but has drawbacks when developing a more complex application, or as part of a Headless Commerce solution.

These drawbacks mainly relate to the organisation and logistical movement of code and content as well difficulties in debugging.
By separating AEM’s management of JavaScript and CSS:

- We achieve the benefits of Headless Commerce, and the concept “Write once, run anywhere”. In addition to these assets being used in an AEM website, they can also be deployed into a UX pattern library, and other web or native applications. With the Adobe solution, you can’t move your library into other platforms, for example, if you are managing a mobile app or legacy site that isn’t in AEM and want to share your code.
- Developers can use the open-source, compilers and application bundlers they know and are comfortable with - i.e. Babel, Webpack, Gulp, Parcel - which allows for superior flexibility in resourcing teams and ensures a reliable speed of development, increasing the time to market for new features and new AEM implementations.
- It’s possible to serve static assets from a number of different, more scalable storage solutions and content delivery networks; Akamai CDN, Amazon S3, etc.
- It’s easier to follow a test-driven development software development process where code is written to meet test expectations. This helps to ensure the stability of an application’s business rules and logic.
- The application architecture is able to adapt to evolving schemes and products, ensuring that upgrade paths are never closed off or require a significant refactoring effort. For example, if code written for one application now has a use case for a non-AEM application, it’s easier to port that code if it’s not coupled to the clientlibs implementation.

In traditional HTL, Adobe’s templating solution, we would reference CSS and JS assets provided in clientlibs for each site component, loading only the code necessary for that component when it exists on the page. But by leveraging Webpack dynamic imports, a custom data attribute mechanism, and the browser’s Mutation Observer API[2], we can use modern techniques to also only load the necessary code while remaining decoupled from the clientlibs setup.
Code Sample 1

Set up your application with Webpack, an open source module bundler, to generate assets that can be deployed outside of a clientlib implementation.

```javascript
// use webpack magic comment to define an application entry chunk file
// https://webpack.js.org/api/module-methods/#magic-comments
function importModule(module) {
  switch (module) {
    case 'component':
      return import(/* webpackChunkName: "component" */ './path/to/component');
    case 'spa':
      return import(/* webpackChunkName: "spa" */ './path/to/spa');
    // etc...
    default:
      console.warn(`${module} does not exist`);
      return Promise.resolve({default: {}});
  }
}

// execute when dom ready
function onDomReady(scope) {
  // Use HTML data attr on markup rendered by AEM component that requires unique JS/CSS module to be loaded
  const dataAemUi = 'data-aem-ui';
  if (!scope) scope = document.body;
  const nodeList = scope.querySelectorAll(`[${dataAemUi}]`);
  nodeList.forEach(domEl => importModule(domEl.getAttribute(dataAemUi))
    .then(module => {
      // dynamic import will wrap module in a 'default' object
      // create an initialize method for your module
      // pass in the reference to the domEl and any other application
      data as arguments
      module.default.initialize(domEl, ...);
    }).catch(err => console.warn(err.message));
  )

  // include code to watch for components added after initial page load
  const observer = new MutationObserver(mutationsList => {
    mutationsList.forEach(mutation => {
      if (mutation.type === 'childList') mutation.addedNodes.forEach(addedNode =>
        onDomReady(addedNode));
    });
    observer.observe(document.getElementById('parsys_wrapper'), {childList: true, subtree: true});
  });
```
Adobe has suggested using solutions like Webpack to generate individual file bundles that map to AEM components via a clientlibs configuration file[3]. However, as Webpack supports file chunking, tree shaking, and dynamic imports, we’re able to ensure that only the code needed for a page component is loaded into the user experience without needing to rely on AEM clientlibs to manage that process.

3. Determine how to plug in widgets or a SPA into your AEM site

The ability to drag and drop a component into any page from an easily accessible list of components is one of the most powerful aspects of AEM. We can leverage the feature to inject JavaScript driven components – individual widgets or full-fledged SPAs – into any page served by AEM.

For simple components, like an accordion or carousel, all of the HTML markup might be provided via an AEM HTL template. For more dynamic components, or SPAs, such as reservation or purchase funnels, this component could consist of nothing more than an empty <div> element.

However, in AEM 6.4 SP2, combined with Adobe built node modules, and the introduction of a node server side by side with a publisher instance, this empty div can be JS server side rendered markup. This is explained below.

4. Understand how to leverage AEM to provide data to a SPA experience

Even if AEM renders as little as a single div element into the browser as a means to inject a more robust and feature-rich, JavaScript-driven experience into the site, we can still leverage AEM’s authoring dialogs & OSGi configurations to provide content, data, and feature settings to the JavaScript/SPA layer. There are several different ways to do this; as part of a JSON object rendered as part of HTL templates, via a headless CMS approach (aka Content as a Service or “CaaS”), or through an integrated, authorable experience now available in AEM 6.4 SP2.

The option you choose should be driven by how tightly you require the authoring experience to be integrated into the end user experience.
If your marketing needs require that content is available outside of AEM, newer versions of AEM allow for page content to be accessible via model.json file extensions so content can be requested and consumed by any type of application; a SPA, another website, native applications, or any IoT device. See JSON Exporter for Content Services[4].

If you require a tightly integrated authoring experience, where AEM renders full-fledged HTML output, and there is no need for externally accessible data, implement a non-CaaS solution where AEM provides data to components via JSON rendered directly into the .html page output. This can be done by including AEM data & content as a JavaScript object in the base page template, or for each component as its added to the page, via an HTML element’s data attribute, or again by using a JavaScript object added to the browser’s window object.
Code Sample 2

Use HTL to provide content and data to components:

```html
<!--/* Retrieve component data and set to a variable */-->
<sly data-sly-use.componentData="${'com.path.to.componentData'}"
    data-sly-unwrap />

<!--/* Add a hard-coded JavaScript object to browser window object */-->
<script>
    // define a namespace object
    window.componentNamespace = {};  
    // retrieve an entire object from AEM
    const componentData = ${componentData.json & context = 'unsafe'};
    // either provide the entire object or translate per JavaScript
    application expectations
    window.componentNamespace.keyName = ${componentData.key_name &
       context="scriptString"};
    window.componentNamespace.isValid = ${componentData.is_valid &
       context="scriptToken"};
</script>
<div class="component">
    <!-- Additional HTML markup -->
</div>

<!--/* Use a HTML data attribute for a component */-->
<div class="component" data-component="${componentData.json & context = 'unsafe'}"></div>

<!-- Additional HTML markup -->
```
5. Leverage AEM 6.4 SP2 to create & manage SPA content via the authoring experience

As mentioned earlier, along with the release of AEM 6.4 SP2, Adobe released a series of node modules that allow for a contextual awareness between a SPA and the authoring experience.

Where previously, via CaaS architecture, JavaScript applications could only know about AEM, this latest release allows for a two-way connection. AEM can become aware of elements within the SPA so an author can visually manage content within the SPA to add/edit components or create new pages while in the authoring mode.

This lets authors create and manage templates in the same context that their users will experience it. Authors can include components as part of a SPA and see them as they drag-n-drop them onto the page within the authoring mode. The components can be built with React instead of HTL. This is instead of a pre 6.4 implementation where the templates of a SPA are “locked” and it would require a code release to change the format/layout within the SPA and only the content/copy would have been authorable.

The two most important node modules that allow for this two-way awareness are:

- [@adobe/cq-spa-page-model-manager][5]: Provides a React application with the page and component structure of an AEM SPA.
- [@adobe/cq-react-editable-components][6]: Maps a React component properties to a a data model resource type authored within AEM.

While AEM still supports pure CaaS models where content is retrieved via URLs and returned in a JSON format, this new feature far exceeds that model in how tightly integrated the authoring experience is. In a CaaS architecture, both AEM and the SPA have to create new pages independently, and potentially via separate production deployments. In this new model, AEM can create entirely new pages in a SPA as a content update outside of production deployments.

This is an extremely powerful step forward in SPA authoring and management for AEM, or any CMS for that matter.

Visit the ‘SPA Getting Started - React’[7] guide for more details on how to develop a SPA in this context.
6. Enhance SPA Architecture to meet SEO Requirements

To satisfy SEO requirements, it's possible to achieve a server-side rendered experience by integrating AEM with an external NodeJS server. Instead of inserting empty <div> elements in the initial page load, an AEM publisher instance supplies authored content to the Node server. That server's code contains references to the same React components used by the SPA, thereby allowing the NodeJS server to use the Adobe-built node modules to render the same markup the SPA would. This is accomplished by utilizing React's server-side rendering (SSR) methods to return HTML to AEM. That markup is then rendered on the initial AEM page load, making it accessible by search engine crawling algorithms. That markup is then immediately replaced by the code powering the SPA.

7. Determine the configurations to a SPA using custom OSGi settings

As much as possible, an SPA should be agnostic in regards to configurations, because the requirements can change across environment modes, languages & locales, and default application settings. In some instances, those configurations could be retrieved via an external service. However, we can leverage AEM custom OSGi configurations so configurable information can be provided to a SPA, and can be updated across environments without a full redeploy of the AEM application.

This OSGi Implementation code is extremely useful for having a SPA work with different APIs depending on the AEM run mode or environment stack. We can point to development versions of an API, and use non-production API keys, before a service is available in a production environment. We can also provide data to the SPA that can trigger additional functionality such as turning site-wide features on and off, redirecting a user from a particular region to a site that is tailored to that region's business needs, or abstracting default application settings that can differ depending on the domain associated with an AEM application.
Code Sample 3

Use OSGi configurations to provide settings to an application:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:osgi="http://osgi.org/jcr/1.0"
  jcr:primaryType="osgi:config">
  apiUrl="https://www.mypiurl.com"
  apiKey="12345abcxyz"
</jcr:root>
```

```java
package com.app.osgi;

import com.app.config.ApplicationConfiguration;
import com.app.config.ConfigurationService;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Property;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.metatype.annotations.Designate;

@Component(servlet = ApplicationConfiguration.class, immediate = true)
public class ConfigurationOsgiService implements ConfigurationService {

  @Activate
  private ApplicationConfiguration applicationConfig;

  @Override
  public ApplicationConfiguration applicationConfig() {
    return applicationConfig;
  }

  @Reference
  private ConfigurationService configurationService;

  // reference that implementation
  @Component(service = ConfigurationService.class, immediate = true)
  public interface ApplicationConfiguration {
    @AttributeDefinition(type = AttributeDefinition.STRING)
    String apiUrl();
    @AttributeDefinition(type = AttributeDefinition.STRING)
    String apiKey();
  }

  <!-- finally, use in HTML -->
  <lib data-sly-use.app="com.app.config.ApplicationConfiguration">
    data-sly-unwrap />
  </script>
  console.log('$(app.apiUrl @ context = "scriptString")');
  // 'https://www.mypiurl.com'
  console.log('$(app.apikey @ context = "scriptString")');
  // '12345abcxyz'
</script>
```
What other factors should be considered when defining SPA architecture?

So far, we’ve looked at the technical architectural decisions involved with integrating a SPA into a single Headless Commerce experience. There are several other factors to consider when defining SPA architecture.

1. Consider Using an API Tier in front of a SPA

While some AEM SPAs only serve content stored within AEM itself, most SPAs will have to integrate data from multiple sources. Rather than relying on JavaScript - that runs in the browser to manage data from various SaaS APIs - it often makes sense to include a unified API management solution in your architecture to orchestrate requests, decorate responses, and extend an application’s features and functionality. As a result, an application can respond to business needs and requirements in front of legacy systems with greater velocity.

Figure 3: An API Tier when integrated into an AEM SPA architecture:

AEM + SPA + API Tier: Requests made to an API Tier then reach out to a varied API Ecosystem to retrieve additional data in JSON format and return that back to the SPA.
In your architecture, these API requests can be managed by AEM servlets, or an external API tier. That API tier can be built with SaaS products such as Apigee and Mulesoft, or via a custom solution. There’s a few benefits to using either a custom or product based API tier instead of relying on AEM servlets as outlined in the following diagram. However, the most important benefit to using an external API tier as opposed to AEM making service requests, is in how it mitigates risk and addresses security concerns. Being able to deploy pieces of your application independently is critical when considering security and stability issues. It is critical to apply software patches and service pack technology upgrades to to safeguard business needs and customer data as quickly as possible when a security risk has been identified. A decoupled API tier plays an essential role in ensuring applications stay current with the latest versions of commerce products and library dependencies.

2. Serve personalized content by using Adobe Target as part of your SPA

One of the most compelling features of brand experiences today is that they can be tailored to the customer. The Adobe Target product suite enables marketers to segment users according to custom rules, and augmenting the experience loaded by the server via its JavaScript library. While this is a dynamic product, there are several challenges to successfully implement it into SPAs. There can be a potential flash of updated content, and it's sometimes necessary to replace portions of the DOM tree to effectively tailor the experience for a user, thereby breaking the React component tree.

However, Adobe has released a React component that can be rendered according to rules managed by Adobe Target[10]. By building our AEM SPAs with React, we can use this library to preserve the React component tree structure and data dependencies, and reduce the amount of code that needs to be written for each Target implementation.

3. Allocate financial budgets judiciously to utilize the expansive Adobe ecosystem

In addition to the architectural benefits of using an API tier, there are cost benefits as well. AEM application servers are required to deliver the things AEM is most specifically designed for, and while you can use them to manage an API tier, there are a few external applications that perform similar functions.
By allocating resources and budgets wisely, businesses can invest more heavily in the broader Adobe Marketing Cloud ecosystem; i.e. Target for personalization, Campaign for promotions, Sensei for artificial intelligence and machine learning, Launch for analytics, etc. The return on investment on those tools far exceeds what is gained by solely using AEM as an API tier.

4. Play to your team’s strengths to maintain development velocity

Content management systems like AEM continue to evolve and offer more robust out-of-the-box SPA solutions.

This effort is largely driven by wanting to provide developers with the tools and languages they are most comfortable with to build content-driven experiences. In a traditional, non-SPA CMS implementation, the learning curve is high for developers who are unfamiliar with the complex parts and patterns of a full-featured AEM application, whether they’re traditional Java or JavaScript developers.

By leveraging either out-of-the-box SPA integrated solutions, or by designing a custom architecture, development teams can take advantage of the the expertise of several AEM developers, and augment that traditional AEM team with developers who feel comfortable working with SPA frameworks like React or Angular. This lets developers use proven, open-source technologies within proprietary applications like AEM to ensure development deadlines are met, teams maintain a rapid release cycle, and can continue to staff and hire smart developers without needing everyone of them to be fully trained on the AEM platform.
5. How to integrate SPA code as part of the code ecosystem

Figure 4: An ecosystem of code:

We previously explained how by using a custom solution for serving a SPA's assets within AEM, we adhere to the concept of “Write once, run anywhere”, and are able to port UI patterns and business logic rules into multiple applications. This is essentially a microservices API approach applied to SPA UI components and data management.

As native devices are now capable of supporting and executing JavaScript outside of a webview, technology teams can build apps with a framework like React Native[11] to leverage the same code that could be served in an AEM-powered website. We can create libraries specific to business needs – for API configurations, form UIs, analytics, date time formatting, purchase funnel state, etc. - and incorporate those into multiple brands and channels into a rapid release cycle.

By leveraging service workers, web notification APIs, and a shared analytics platform across devices, marketers can track a user's customer journey and communicate with that customer across channels via those shared codebases.

We can even use frameworks like React Native for Web[12] to visualize native app screens as part of a SPA integrated into AEM so authors can manage mobile application content and see what it will look like on a native application. Using tools like MS CodePush[13] we can then take the updated SPA content and deploy it to a native application without a full app store release.
Conclusion: AEM & SPAs in an evolving landscape

As businesses evolve to meet their customers’ needs, commerce platforms continue to evolve to meet the needs of marketers. AEM has made major strides in its last two version updates to support more dynamic CaaS models, and SPA integration directly in the authoring experience - helping to keep the experience fresh and regularly updated.

To take advantage of these upgrades and to continue to evolve customer experiences, it's imperative to architect solutions that enable teams to revisit and upgrade implementations without fully replacing the entire architecture. Doing so mitigates the risk of a single solution becoming outdated or a security vulnerability exposing risk to an entire customer facing application.

For today, and in the future, a mix of custom solutions, integrated with evolving out-of-the-box products, will continue to be the most reliable path forward for marketers and development teams to rapidly deploy features and evolve to meet customer needs and expectations.
Isobar’s AEM case studies

Re-designing the travel experience for Jetstar [14]

Digital has transformed where and how people book travel all over the world. Jetstar, a leading low-cost Australian airline owned by Qantas, needed to ensure it delivered an inspiring customer experience and superior service to stay competitive in a crowded market. Isobar redesigned Jetstar’s global websites and eCommerce platform from the ground up, keeping the customer at the heart of the experience, iteratively testing and rolling out 22 websites in 9 languages across 14 regions. Jetstar.com is now 4 times faster than their competitors and time to book a flight has been reduced by 15%.

Injecting Colour Into Digital for Dulux [15]

DuluxGroup has 30+ brands that cater for both consumer and professional customers and each had a completely different website; some outdated, some quite new but all built on different and unique platforms, which cost a substantial amount to maintain. Isobar was tasked with creating a platform that integrated the different brands into one unified system. Working with our UX, creative and technology teams, we developed a component based design system that is modular, flexible and scalable using Adobe Experience Manager. Website traffic has increased more than 25% and on page engagement, almost doubling since launch.

Empowering HBO’s Success [16]

With a leading entertainment media platform, HBO needed a full multi-channel content delivery redesign to enable production team meet their goals for fan engagement and reach. Isobar provided a strategy for HBO’s digital path and used Adobe Experience Manager and Adobe Target to create rich, intuitive, multi-channel experiences that engage millions of fans. We created a single, consistent and streamlined content creation and publication system along with a digital data and optimization platform that enables HBO content creators and producers to deliver a modern, responsive design and experience.
Glossary

Headless Commerce
A decoupled customer experience from the back-end technology infrastructure enabling brands to deliver a truly customer-centric experience.

Presentation Layer
The touch point where the customer engages with the brand, and relationships are built.

JSON
JavaScript Object Notation is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

API Ecosystem
The technical definition of the communication channels between a brand’s business applications and touch points.

AEM Application Server
In the context of AEM, an application server is the AEM software environment that enables authoring, publishing, and serving content to end users.

Development Velocity
An agile development metric that determines the amount of work a team can complete in a development sprint.

API
An application programming interface is a collection of services used when building software products; usually accessible via HTTP methods, that returns data in a UI agnostic, consumable format (i.e. XML, JSON, etc.)

Application Bundler
A piece of software used to package various modules into deployable artefacts, for the purpose of deploying those artefacts in a browser, web server, or native application.

CaaS
Content as a Service describes a mechanism where an application’s content is decoupled from its presentation layer; usually delivered in a JSON or XML format.
Glossary

CDN
A content delivery network is a system of distributed servers that deliver static resources depending on the geographic locations of the user to reduce the load time of a user experience and burden on application servers.

CMS
A content management system is a software application used to author content for digital platforms and applications.

Compiler
A piece of software used to transform human-friendly programming code into a context more easily understood by another application which consumes the output of the software compiler for its own purposes.

HTL
The HTML Template Language introduced with AEM 6.0 “as the preferred and recommended server-side template system for HTML.

Microservices
An architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable, and organized around business capabilities.

Pattern Library
A visual collection of user experience design patterns that can be applied to various, branded software applications.

React Native
React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI using declarative components.

Refactoring
The process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.
**Glossary**

**SaaS**
Software as a Service describes a software application that is licensed on a subscription basis and centrally hosted outside of the application that leverages the application.

**SEO**
The process of improving a website’s search rankings by making changes to a site’s content structure, implementing configuration files designed to be crawled by search engines, and improving web page load performance metrics.

**SPA**
A Single Page Application takes advantage of modern browser APIs that enable a website to behave more like a native mobile or desktop application.

**SSR**
Server Side Rendering is the process of using JavaScript content retrieval methods on NodeJS servers to render HTML from the server that mimics markup that is subsequently rendered by a browser-based JavaScript application.

**Tree Shaking**
A process by which an application bundler will remove dead end code when compiling a build artefact.
Sources

1. Headless Commerce model
2. Mutation Observer API
3. Clientlibs configuration file
4. JSON Exporter for Content Services
5. @adobe/cq-spa-page-model-manager
6. @adobe/cq-react-editable-components
7. SPA Getting Started - React
8. React’s server-side rendering (SSR) methods
9. OSGi configurations
10. Rules managed by Adobe Target
11. React Native
12. React Native for Web
13. MS CodePush
14. Re-designing the travel experience for Jetstar
15. Injecting Colour Into Digital for Dulux
16. Empowering HBO’s Success
About Isobar

We are a global digital agency transforming businesses and brands through the creative use of digital. Our 6,500 digital experts in 85 locations across 45 markets in Americas, EMEA and APAC deliver experience-led transformation, powered by creativity through our end to end service offering.

Isobar’s clients include Adidas, Coca-Cola, Enterprise, KFC, Mead Johnson, Nestle and Philips, and is part of Dentsu Aegis Network, a wholly owned subsidiary of Dentsu Inc. www.isobar.com.

Key contacts

Vikalp Tandon, Chief Technology Officer, Isobar Global
vikalp.tandon@isobar.com

Hannah Ainsworth, Marketing Director, Isobar Global
hannah.ainsworth@isobar.com

Brian Fegan, Technical Director, Isobar US
brian.fegan@isobar.com