VIEW POINT

HEADLESS WITH SITECORE JAVASCRIPT SERVICES (JSS)

Abstract

Digital content is becoming key to engage audiences across all industry domains. However, as customer touchpoints rapidly increase, marketing teams are challenged with several technological and operational challenges. This paper looks at headless architecture as a solution that enables massive flexibility by decoupling authoring and publishing capabilities, and driving the consumption of content at scale. It also provides a real-world case study of a Sitecore JSS implementation by Infosys along with key recommendations for digital content marketing teams.

Infosys®
Navigate your next

Introduction

Enterprises are viewing the importance of 'content' as more than informational relevance. Content is now a powerful tool that drives accurate marketing and enhances personalization, creating a powerful network of influencers. The constant proliferation of digital consumer touchpoints and the increasing need for channel-specific delivery and content variance have made it necessary for digital content management teams to quickly address technological and operational challenges. This is difficult as marketers handle disparate marketing content in silos across systems that must be curated to channel-specific needs.

Headless architecture is emerging as a solution to address these challenges.

What is Headless Architecture?

Headless architecture is a specialization of decoupled architecture where presentation layer of an application is separated from its backend services. In the digital marketing domain, this architectural paradigm polymorphs as a headless content management system (CMS), or Content-as-a-Service.

Content is managed in a centralized platform and delivered to disparate channels through application programming interfaces (APIs). Headless architecture provides flexibility to decouple CMS platforms from channels. It also modernizes content strategies either in parts or in a big bang way without impacting channels. Further, it equips designers with the freedom to create marketing assets without being constrained by backend technologies, including the CMS. On the cost side, it helps reduce maintenance involved in multi-site/multi-app content scenarios.

Sitecore's Offerings for Headless Architecture

Sitecore has emerged as the pioneer of the item-based headless CMS and offers multiple ways to build headless applications. Digital content users can leverage any one of the following software development kits (SDK) or accelerators according to their business needs:

- Sitecore JSS using JavaScript SDK – This helps implement headless architecture using modern JavaScript technologies like React, Angular, or Vue. JavaScript developers need not necessarily be trained in Sitecore. The entire implementation can be done at the front-end and deployed later to Sitecore to create the necessary artefacts in Sitecore. This also enables users to develop applications using modern technology without sacrificing CMS capabilities and advanced marketing features.
- .NET Core SDK – This is another type of headless architecture implementation that accelerates website development by using the latest .NET technology. This capability is provided by Sitecore for Sitecore 10 version onwards.
- Next.js SDK – This is a cutting-edge feature provided by Sitecore. It improves performance, reduces the load on backend servers, and ensures client-side application stability. It simplifies JSS development by supporting internationalization, out-of-the-box (OOB) Server-side rendering (SSR), TypeScript, environment-level variable management, performance metrics, image optimization, component-level cascading style sheet (CSS) management, lazy loading, etc. Static site generation (SSG) is a significant feature provided by Next.js that pre-renders the app rendering code thereby reducing or eliminating the time spent to process the app

rendering code when the end-user interacts with a page.

- Sitecore Experience Accelerator (SXA) – Headless architecture can also be implemented using SXA. Sitecore provides JavaScript Object Notation (JSON) artefacts such as JSON layout, JSON renderings, and JSON variants. These should be used to build the page so that the SXA layout service can render the data in JSON format.

Why Choose Sitecore JSS

1. It supports headless and hybrid headless architecture

Sitecore JSS provides capabilities to build websites using headless and hybrid headless architecture.

If modern JavaScript frameworks like React, Angular, etc., are being used to develop the presentation components due to their lightweight and high-performing nature and content is consumed through APIs, then the content management capabilities are compromised. Sitecore JSS bridges this gap. It enables developers to build components using modern JavaScript frameworks like React, Angular, Vue, or Next.js while retaining the benefits of CMS and advanced marketing capabilities.

Sitecore JSS is useful when a website is transactional or dynamic in nature with several upstream and downstream transactions and when all the features of CMS and advanced marketing capabilities are needed. Additionally, Sitecore supports the co-existence of headless JSS apps as well as traditional .NET Core or Sitecore model-view-controller (MVC) apps within the same application in the same instance. Sitecore leverages a hybrid approach and allows enterprises to migrate to JSS at their own convenience. It offers the flexibility of building a few parts of the application in MVC and other parts in JSS as per the requirement and design.

2. It provisions Sitecore experience platform capabilities

'Experience editing' and 'preview' are vital features of a CMS. In a traditional headless architecture implementation, these features are compromised.

JSS brings in the same content authoring and editing experience as traditional Sitecore. Node Package Manager (NPM) packages include the necessary techniques to support experience editing. It enables authors to preview the page in device simulators before publishing. JSS also offers a robust experience platform covering analytics, personalization, and A/B testing as described below:

- Analytics – Every 'layout service call' for the route is considered as a 'page view' of the route. Page views and page events can be triggered using the JSS Analytics service. Experience analytics and path analyzer reports are supported in JSS.
- Personalization – It enables content authors and marketers to personalize JSS components and understand the reach of each component
- A/B testing – A/B testing can be performed on the variants of the JSS components similar to how it is done on traditional Sitecore components. A complete page can undergo A/B testing with another page or previous version of the same page.

3. It provides extensive documentation

Detailed documentation is available for the developers to start using the SDK. Sitecore provisions a quick start kit and sample apps in React, Angular, Vue, Next.js, and .NET Core. Sitecore communities like Sitecore Stack Exchange and Sitecore Slack Community are useful resources for developers looking for help if they are stuck with an issue.

JSS Reference Architecture

JSS is constructed with several APIs and services as shown in Figure 1. A Sitecore page is called as a route in JSS and all the components and data of the routes are defined dynamically by Sitecore to support data-driven personalization and A/B testing.

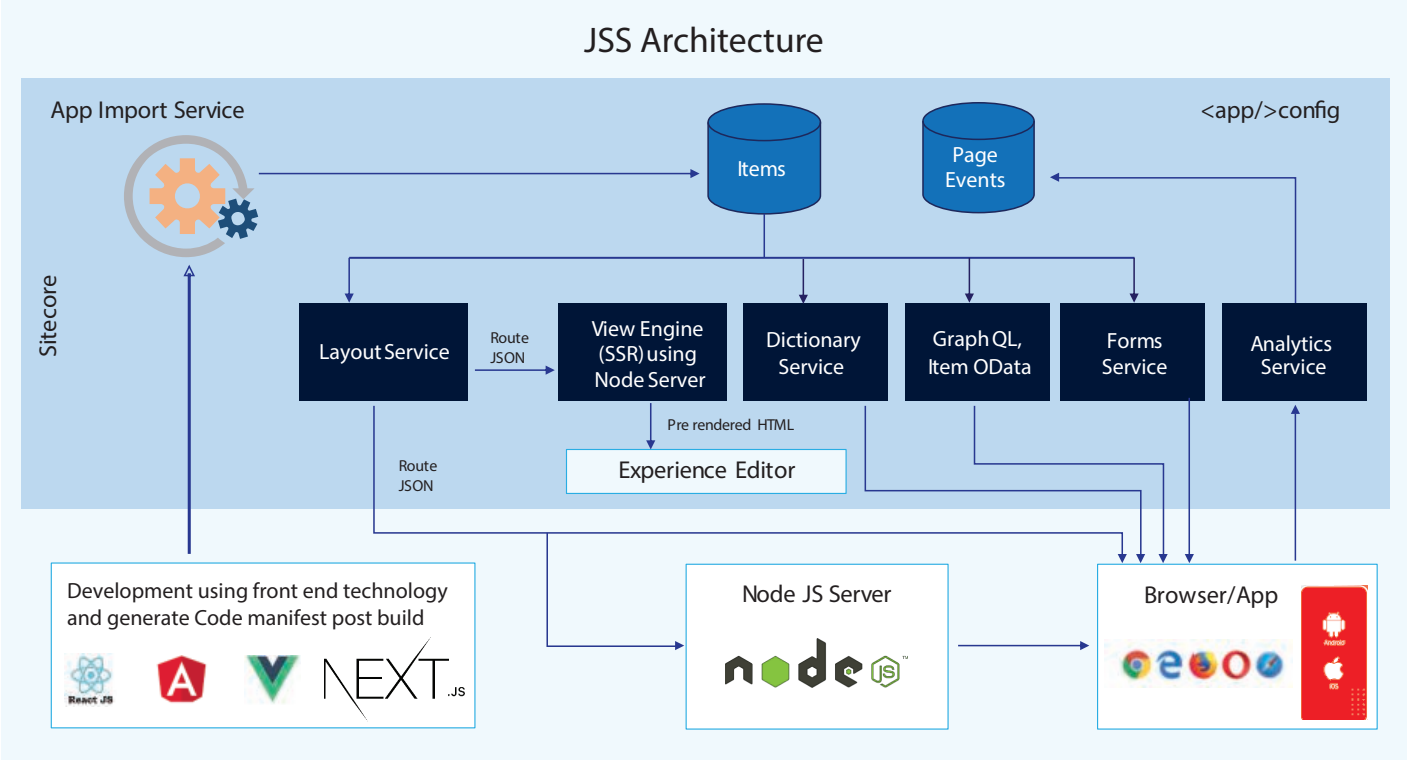


Figure 1 – JSS refence architecture

The key JSS elements are:

- A JSS library that includes a wide range of NPM packages. It facilitates working with Sitecore data and layouts in JSS.
- A layout service that exposes the component details and the data for each component
- A JavaScript view engine that enables Sitecore to perform SSR of JavaScript apps for experience editing
- An app import feature that enables Sitecore to import the JSS app into Sitecore and create the necessary Sitecore artefacts (layouts, templates, placeholders, items, etc.)

How to Get Started with Sitecore JSS

JSS offers various workflows and several modes of working. A few configuration changes and server set-up must be done to activate JSS. These are explained in detail below:

1. **Choose developer workflow**

   Sitecore JSS offers two types of developer workflows to get started with JSS. These are:

   - Code-first workflow – This workflow helps developers work with mock content in the 'disconnected mode' without connecting to the Sitecore instance. Front-end developers can create a manifest of layouts, templates, items, fields, etc., using client SDKs and file structure. The application import pipeline is used to import the JSS app to Sitecore and create the necessary artefacts. This approach is recommended when the application has a simple content structure and contains more dynamic data.
   - Sitecore-first workflow – This is similar to building a non-JSS Sitecore site. Developers must use the JSS-built base templates instead of Sitecore base templates. For example, JSON rendering needs to be used instead of view/controller/ extensible Stylesheet Language Transformations (XSLT) rendering. This workflow is recommended when the application has a complex content structure.

2. **Set up the JSS server**

   Setting up the JSS server is necessary to import the JSS app into the Sitecore instance. The app works in connected mode and the SSR works only if the JSS server is set up. Some key requirements to set up the JSS app are:

   - Procure a JSS-enabled Sitecore license for the Sitecore instance
   - There should be a node server running in the Sitecore instance for the JSS app
   - Create an API key in Sitecore in the 'Services' folder
   - Install the Headless Server Components Sitecore package from the official JSS website in order to unlock the server components

3. **Configure the apps**

   The Sitecore instance must know which app should be imported. Thus, configuring the JSS app in Sitecore is a mandatory step to run the app in the 'integrated mode'.

4. **Choose the application mode**

   JSS gives developers the option of choosing various modes of working so they can accelerate and simplify the development process. These modes are:

   - Disconnected mode – Developers can work with local mock content without a Sitecore instance for faster development
   - Connected mode – Developers can work in the system's local host in connection with Sitecore by fetching the data from Sitecore
   - Integrated mode – Developers can work in the Sitecore instance URL in connection with Sitecore by fetching data from Sitecore
   - API-only mode – Any platform that understands JSON can consume the JSS APIs and personalized layout information

5. **Implement analytics**

   Every layout service call is counted as a page view in Sitecore. To disable page view tracking of the layout service, developers must set the parameter "tracking=false" and send this to the service as an additional parameter. If tracking is deliberately disabled, then A/B testing and personalization reports will not be generated. Thus, the tracking parameter should be disabled only when there is no requirement of analytics. This decision must be made judiciously. The analytics tracking API allows firing events, goals, outcomes, campaigns, and page/route views from the JSS app.

6. **Deploy the application**

   There are various JSS commands available to deploy the JSS app into Sitecore. Ex- "JSS deploy app" deploys files and items of the app to Sitecore, "JSS deploy app -c -d" deploys files and items along with the content and dictionary, there are several other commands available which can be used as per your app requirement.

7. **Hosting**

   JSS fits into all the deployment models supported by Sitecore. Hence, it can be deployed to Platform-as-a-Service (PaaS) servers for enterprises with Sitecore 9.x versions as well as containers for those with Sitecore 10.x versions.

8. **Other considerations**

   There are other useful concepts in JSS to be considered during development as per the requirements. These are:

   - GraphQL support – This is a powerful concept in JSS. With GraphQL queries, the format of the route data returned for a specific component can be modified. Users can query the data source and retrieve the data as needed from the layout service. JSON renderings have a GraphQL editor that helps write the queries
   - Forms service – It can consume and post Sitecore forms from JSS apps
   - Dictionary service – JSS SDK has a simple API to utilize the dictionary service
   - Support for existing MVC components in a JSS app – Sitecore MVC components and JSS components can coexist in an application. If there are existing static MVC components such as header or footer, those can be leveraged in the JSS application instead of creating new ones. To do this, a simple step should be performed on the static component, i.e., mark the attribute "json=false" in the MVC rendering. An important consideration here is that the front-end code should be dynamic so as to pull the data from the MVC renderings.

AN INFOSYS CASE STUDY

Utilities service provider leverages Sitecore JSS for stronger analytics on dynamic webpages

Background

Infosys collaborated with a leading power service provider to build a public customer-facing website. The website had a few dynamic self-service pages for actions like making a payment, starting a new service, terminating a service, etc. It also hosted several information-rich static pages. The website was heavily used by millions of end-users.

The static pages rendered content from Sitecore. However, in the self-service dynamic pages, the user was presented with a multi-step form to collect inputs. In the last step, the user had to submit the form. Multiple API calls were made to perform upstream and downstream transactions on these dynamic pages. As users proceeded through the multiple steps of filling a form on a dynamic page, the URL of the page did not change, similar to a single page application (SPA) implementation.

Infosys leveraged hybrid architecture to execute a headless architecture Sitecore implementation. Static pages were built using Sitecore MVC. Since multiple API calls were needed for the dynamic pages, React was chosen to build the front-end presentation layer, fetch transactional data, and improve website performance. The content for the dynamic pages remained in Sitecore and was served using a custom API. The container for dynamic pages were Sitecore pages with React components.

Challenges

Sitecore Analytics was used to extract reports of the application users. A key outcome here was that analytics reports for the static pages were displayed within the Experience Analytics, Path Analyzer, and Experience Profile modules. However, analytics data for dynamic pages was scarce. The client wanted in-depth analytics reports for dynamic pages in order to understand exactly at which step did users abandon the form, how many users reached a particular step on the form, how to personalize React components, what is the level of A/B testing on React components, etc. Such insights could not be supported by a traditional headless architecture approach where React was at the front-end, consuming Sitecore content through APIs.

Another major challenge with the dynamic pages was that content authors could not 'experience edit' the page or preview the page before publishing. The reason for this is that content was being created in Sitecore as labels and exposed through APIs. There was no structured content for the dynamic pages in Sitecore. A few workarounds were recommended to set up the Experience Editor by adding renderings with dynamic placeholders and sourcing the data of each label in the rendering. However, these were effort intensive and complex.

Infosys Solution

Sitecore JSS offered several critical features that would address the requirements of the utility company with respect to its dynamic webpages, experience editing, previewing, analytics, personalization, and A/B testing. It also offered all CMS and advanced marketing capabilities for dynamic pages.

Infosys conducted a proof of concept (PoC) in JSS 12.0 for a single self-service dynamic page where the container was a Sitecore page and each step was created as a route in JSS. The layout service in JSS pulls data from the routes. Every time the layout service is called for a particular route, the page view of the route increases by 1. In this way, analytics was also established.

Infosys chose to implement a hybrid headless architecture model with JSS primarily because the dynamic page container was still a Sitecore page with React JSS components. JSS supported the creation of templates, data sources, and renderings on the client-side using NPM packages. Thus, content was structured, enabling experience editing, previewing, and device previewing of a page. Various components were created to increase modularity, thereby allowing Infosys to perform A/B testing and personalization. Sitecore JSS was eventually used for all the dynamic pages. These pages could also leverage the CMS and advanced marketing capabilities provided by Sitecore.

Infosys Notes

- In the code-first workflow, the data schema and content are developed in the front-end in JavaScript and YAML Ain't Markup Language (YAML) files. Once this manifest is deployed to Sitecore using the import pipeline, there is no reverse-sync mechanism to push data from Sitecore to the JSS app. Thus, Infosys chose to shift from a code-first approach to a Sitecore-first approach for the already developed components once the app was deployed to production.
- There is no OOB feature to deploy the JSS manifest into Sitecore at the desired location for the benefit of content authors. It will always deploy to the default location.
- In the layout service calls, if tracking is disabled for a route, then the page view does not increase for the route. Additionally, A/B testing and personalization reports are not generated. Thus, it is important to be judicious when creating the number of routes and disabling tracking for routes.

KEY TAKEAWAYS

Advantages of Sitecore JSS

- It enables headless and hybrid headless architecture implementation by retaining content management and advanced marketing features
- It empowers JavaScript developers to implement CMS-based applications using cutting-edge technologies
- JSS and SXA can coexist in an application that supports hybrid models

Challenges of Sitecore JSS

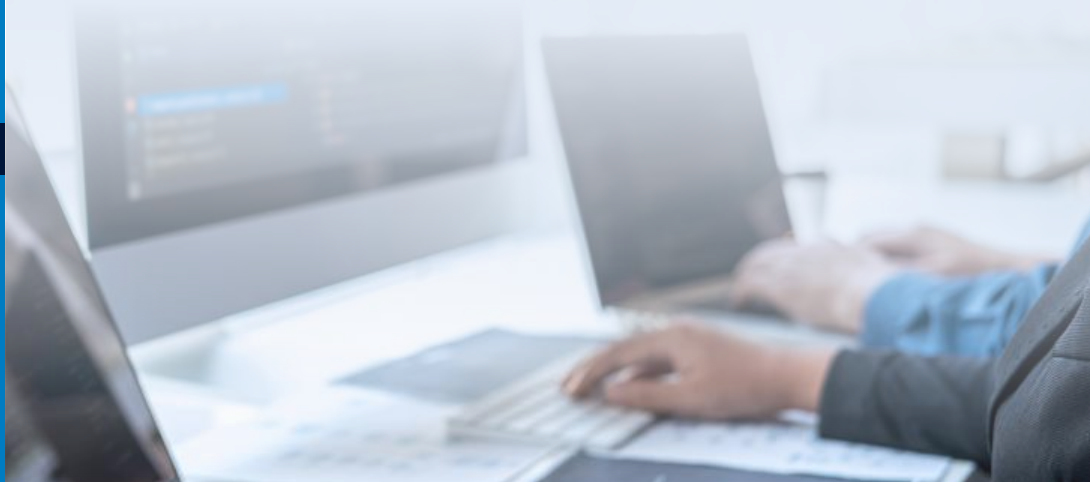
- Any implementation at the foundation layer or at the application start layer still resides on the server-side. For example, if a pipeline needs to be tweaked, then it has to be done on the server-side.

About the Infosys Sitecore Practice

Infosys has extensive experience implementing and delivering successful digital transformation programs to multiple customers across geographies and business domains using Sitecore JSS.

Infosys has wide and deep expertise in pure headless and hybrid headless architecture implementations using JSS. These implementations cover all JSS features such as Experience Editor, device preview, personalization, A/B testing, firing page events, and Path Analyzer. Infosys has worked extensively with Sitecore on the analytics aspects of JSS and has a deep understanding of its features.

Infosys is a Sitecore implementation partner with rich experience in transforming over 200 customer experience channels. The Infosys Sitecore Practice covers the entire spectrum from websites/apps development and multilingual implementations to technology migration to Sitecore, Sitecore version upgrades, and migrating customer experience solutions to cloud.



About the Author

Shakti Patro

Lead Consultant, Digital Marketing, Infosys

Shakti is a Lead Consultant for Digital Marketing in Infosys. She has 13 years of IT and consulting experience with expertise in architecture definition, design, and development across the Microsoft application stack as well as web content management systems. She is Sitecore-certified and has architected several Sitecore-based marketing and dynamic applications. Shakti has effectively led multiple projects and provided innovative solutions for clients to suit their business needs.

For more information, contact askus@infosys.com

Navigate your next