CONSIDERACIONES PARA LA IMPLEMENTACIÓN DE COMPONENTES DE CONSULTA OPENSEARCH Y DSPACE API EN CMS ESTÁTICOS Y CMS SIN CABECERA

Villarreal Gonzalo Lujan
Universidad Nacional de La Plata PREBI-SEDICI;
Comisión de Investigaciones Científicas CESGI
gonzalo@prebi.unlp.edu.ar

Lautaro Josin Saller
Universidad Nacional de La Plata PREBI-SEDICI;
lautaro.josin@sedici.unlp.edu.ar

Mazullo Juan Cruz
Universidad Nacional de La Plata PREBI-SEDICI
juan.mazullo@sedici.unlp.edu.ar

Carlos Nusch
Universidad Nacional de La Plata PREBI-SEDICI y
Comisión de Investigaciones Científicas CESGI
belchior@fiocruz.br

DOI: *10.22477/xiv.biredial.377*
EJE TEMÁICO: *Infraestructura tecnológica*

RESUMEN

Este trabajo analiza los desafíos y oportunidades de adaptar componentes de búsqueda basados en OpenSearch y la API de DSpace a arquitecturas modernas de sitios web, específicamente los generadores de sitios estáticos (SSG) y los CMS sin cabecera. Estos nuevos enfoques ofrecen mejoras en rendimiento, seguridad y escalabilidad respecto a los CMS tradicionales como WordPress o Joomla, pero requieren un replanteo técnico y conceptual profundo. El trabajo expone cómo estas arquitecturas desacopladas implican repensar la estructura de los componentes de consulta existentes, tanto en cuanto a las herramientas tecnológicas utilizadas como a los paradigmas sobre los que fueron originalmente diseñados. Se presentan estrategias como el uso de índices de búsqueda preconstruidos, funciones serverless o el patrón Backend for Frontend (BFF), considerando también aspectos críticos como la seguridad, la gestión de credenciales y el rendimiento. Finalmente, se destaca que la correcta integración de estos componentes en CMS modernos es clave para garantizar el acceso eficiente y seguro a la información académica en repositorios institucionales.

Palabras-clave: Sitios web institucionales ; Repositorios institucionales ; Publicaciones científicas; Centros de investigación.

ABSTRACT

This paper analyzes the challenges and opportunities of adapting search components based on OpenSearch and the DSpace API to modern website architectures, specifically static site generators (SSG) and headless CMS platforms. These new approaches offer improvements in performance, security, and scalability compared to traditional CMSs like WordPress or Joomla, but require a deep technical and conceptual rethinking. The study explores how these decoupled architectures demand a reconsideration of the structure of existing query components, both in terms of the technological tools used and the paradigms on which they were originally designed. Strategies such

as the use of prebuilt search indexes, serverless functions, or the Backend for Frontend (BFF) pattern are presented, also taking into account critical aspects such as security, credential management, and performance. Finally, the paper highlights that proper integration of these components into modern CMS platforms is key to ensuring efficient and secure access to academic information in institutional repositories.

Keywords: Institutional websites; Institutional repositories; Scientific publications; Research centers.

INTRODUCCIÓN: CONTEXTUALIZACIÓN DE LOS REPOSITORIOS INSTITUCIONALES Y LA EVOLUCIÓN DE LAS ARQUITECTURAS CMS

Los repositorios institucionales (RIs) desempeñan un papel fundamental en la preservación y difusión de la producción académica, actuando como plataformas esenciales para el acceso a datos de investigación, publicaciones y otros recursos académicos (De Giusti, 2022). Históricamente, los sistemas de gestión de contenidos (CMS) tradicionales y dinámicos, como WordPress y Joomla, y Choique en el caso particular de la Universidad Nacional de La Plata, han sido utilizados para construir sitios web institucionales e integrar estos RIs (Villarreal, Manzur, Vila, & De Giusti, 2017). La arquitectura típica de estos CMS se caracteriza por una capa de presentación frontal estrechamente acoplada con la gestión de contenido y la base de datos en el *backend*.[1] En este contexto, se implementó un componente en PHP para facilitar las consultas a través del protocolo OpenSearch y la API de DSpace, funcionando como *plugins* dentro de estos sistemas (Villarreal, Salamone Lacunza, Vila, De Giusti, & Manzur, 2017).

Sin embargo, en el panorama actual del desarrollo web, han surgido alternativas modernas como los generadores de sitios estáticos (SSGs) y los CMS sin cabecera (*headless*). A diferencia de los CMS tradicionales, cuya lógica se ejecuta en el servidor en tiempo real, los SSGs pre-construyen archivos HTML estáticos, mientras que los CMS sin cabecera separan el repositorio de contenido del *frontend*, entregando el contenido a través de APIs. La adopción de estas nuevas arquitecturas responde a la necesidad de mejorar el rendimiento, la seguridad, la escalabilidad y reducir los requerimientos de recursos de hardware (Gustafsson, 2019).

En este contexto de evolución arquitectónica, los componentes de consulta OpenSearch y DSpace API, originalmente diseñados para el entorno dinámico de los CMS tradicionales, pueden no ser directamente compatibles o funcionar de manera óptima dentro de las arquitecturas estáticas y basadas en API de los SSGs y CMS sin cabecera. Por lo tanto, el objetivo de este artículo es evaluar las consideraciones y estrategias necesarias para adaptar estos componentes al nuevo panorama, manteniendo su funcionalidad y alineándose con la filosofía subyacente de los SSGs y CMS sin cabecera.

La tendencia hacia los SSGs y CMS sin cabecera representa una evolución arquitectónica significativa en el desarrollo web para sitios institucionales (centros de investigación, laboratorios, departamentos, *etc.*) (Diaz, 2018). Esta evolución está impulsada por la necesidad de un mejor rendimiento y escalabilidad, cruciales para manejar la creciente cantidad de contenido

académico y el tráfico de usuarios a los RIs.

Cabe aquí realizar una breve caracterización del contenido que usualmente se publica en sitios web institucionales. Este tipo de espacios publica típicamente información sobre la institución, como ser sus objetivos e historia, líneas de investigación, proyectos activos, personal involucrado e información de contacto. Este tipo de información suele cambiar de manera esporádica, con lo cual el requerimiento de actualización periódica será reducido. Sin embargo, estos sitios suelen también publicar sus actividades y resultados, ya sea en forma de noticias donde publicitan sus avances, eventos, campañas, *etc*. como también en forma de publicaciones científicas (artículos, trabajos en congresos, tesis, entre otros recursos). El grado de actualización de estos contenidos, ya sean noticias o publicaciones científicas, suele estar fuertemente vinculado al tamaño de cada institución; por ejemplo, un laboratorio con poco personal publicará pocos artículos al año, mientras que un instituto de investigación con cientos o miles de miembros podría publicar actualizaciones y artículos científicos más de una vez por semana. Es por ello que, al momento de definir un CMS para sitios institucionales, debe tenerse en cuenta no solo los tipos de contenido que publica, sino también el grado de actualización de cada uno de ellos, discriminando contenidos estáticos, contenidos de actualización esporádica, y contenidos de actualización contínua o permanente. Finalmente, es importante destacar que en la línea de trabajo aquí presentada, el contenido científico no es gestionado como contenido dentro del mismo CMS, sino que es obtenido desde repositorios institucionales mediante protocolos como OpenSearch o API REST; la Tabla 1 agrupa los diferentes tipos de contenidos según estos criterios.

Tabla 1 - Tipos de contenidos y origen de los mismos en los sitios web institucionales

Contenido estático local	Contenido Dinámico local	Contenido dinámico externo
Historia	Noticias	Artículos en revistas
Objetivos	Campañas	Trabajos en congresos
Líneas de investigación	Congresos y eventos	Tesis y tesinas
Integrantes		Libros
		Datasets

Fuente: los autores

Los CMS tradicionales, aunque inicialmente convenientes, pueden volverse intensivos en recursos y potencialmente más lentos con el aumento de la complejidad y el tráfico. Los SSGs y CMS sin cabecera ofrecen ventajas inherentes en rendimiento al desacoplar la generación y la entrega de contenido (Tramullas, 2020). Esto sugiere una tendencia hacia arquitecturas web más eficientes y especializadas para las instituciones. La inversión y la experiencia existentes del usuario con *plugins* basados en PHP para CMS tradicionales proporcionan una base valiosa, pero también presentan un desafío. La adaptación requerirá comprender las dife-

rencias fundamentales en cómo estos nuevos sistemas manejan la funcionalidad dinámica y la recuperación de datos. Los *plugins* actuales dependen de la ejecución de código PHP en el lado del servidor para interactuar con las APIs de OpenSearch y DSpace, procesar la información obtenida y mostrar los resultados dentro del *framework* del CMS. Los SSGs y CMS sin cabecera a menudo dependen de JavaScript en el lado del cliente o de procesos en tiempo de construcción para tales características dinámicas. Esto implica la necesidad de repensar el enfoque de implementación.

FUNDAMENTOS DE LOS GENERADORES DE SITIOS ESTÁTICOS Y LOS CMS SIN CABECERA: VISIÓN GENERAL ARQUITECTÓNICA Y PRINCIPIOS FUNDAMENTALES

Los generadores de sitios estáticos (SSGs) operan tomando como entrada contenido, típicamente en formatos como Markdown, reStructuredText, JSON o XML, y plantillas, usualmente escritas en HTML con sistemas de templating como Liquid, Go templates o *frameworks* de JavaScript. Durante un proceso de construcción, el SSG pre-renderiza páginas HTML estáticas que luego se despliegan en un servidor web o una red de entrega de contenido (CDN). A menudo, los SSGs se integran con *frameworks* de *frontend* como React, Vue.js y Angular para mejorar la interactividad. Los motores de templating juegan un papel crucial al proporcionar una estructura básica para las páginas web.

El principio fundamental de los SSGs radica en la generación de contenido en tiempo de construcción, lo que resulta en velocidades de carga rápidas ya que no se requiere procesamiento en el servidor para cada solicitud. Este enfoque se alinea con la metodología JAMstack (JavaScript, APIs, Markup), donde los SSGs a menudo constituyen la parte de "Markup", mientras que JavaScript maneja las funcionalidades dinámicas y las APIs conectan con servicios de *backend*. Además, todo el sitio, incluyendo el código y el contenido, puede gestionarse mediante sistemas de control de versiones como Git (Tramullas, 2020). La ausencia de bases de datos y lógica del lado del servidor también reduce la superficie de ataque, mejorando la seguridad.

Por otro lado, un CMS sin cabecera separa el repositorio de contenido *backend* (el "cuerpo") de la capa de presentación *frontend* (la "cabeza"). El contenido se almacena en una base de datos y se entrega como datos, típicamente en formato JSON o XML, a través de APIs (RESTful o GraphQL) a cualquier aplicación *frontend* (Jain, 2021). El *backend* incluye una interfaz editorial para los creadores de contenido.

El principio central de un CMS sin cabecera es el desacoplamiento de la gestión de contenido de la presentación, lo que permite a los desarrolladores utilizar sus tecnologías *frontend* preferidas. El enfoque "API-First" asegura que el contenido se acceda y se entregue a través de APIs, haciéndolo independiente del canal y habilitando estrategias de "crear una vez, publicar en todas partes". Esto facilita la entrega de contenido a diversas plataformas y dispositivos (sitios web, aplicaciones móviles, dispositivos IoT) desde una única fuente de contenido. La arquitectura desacoplada también permite una escalabilidad independiente del *frontend* y el

backend, proporcionando mayor flexibilidad en la elección de tecnologías (Jain, 2021).

Una característica fundamental tanto a los SSGs como a los CMS sin cabecera es la separación de responsabilidades. Los SSGs separan la generación de contenido del servidor, mientras que los CMS sin cabecera separan la gestión de contenido de la presentación. Esta separación es clave para lograr rendimiento y flexibilidad. Al pre-construir páginas (SSGs) o proporcionar datos brutos a través de APIs (CMS sin cabecera), estos sistemas reducen la carga de procesamiento en el servidor durante las solicitudes de los usuarios. Esto conduce a tiempos de respuesta más rápidos y una mejor escalabilidad (Gustafsson, 2019). La separación también permite una mayor innovación y elección en las tecnologías *frontend*. Si bien ambos ofrecen beneficios de rendimiento, atienden a diferentes necesidades. Los SSGs son ideales para sitios web con contenido en gran parte estático, pero pueden requerir procesos de construcción más complejos para las características dinámicas (Newson, 2017). En el contexto de este trabajo, muchos sitios web institucionales poseen esta característica, en especial cuando se trata de centros de investigación con baja tasa de publicación de novedades y actualizaciones. Los CMS sin cabecera son más adecuados para la entrega de contenido complejo y multicanal donde el contenido necesita ser reutilizado en diversas plataformas (Jain, 2021). Este tipo de CMS quizás se adecúe más para grandes institutos de investigación, que dispongan una estructura web jerárquica (muchos subsitios), distribuída (los subsitios se alojan en diferentes servidores) y federada (la gestión y mantenimiento de cada subsitio cae bajo la responsabilidad de diferentes actores de la institución). Sumado a las características de cada institución, la elección entre un SSG y un CMS sin cabecera también debe considerar los requisitos específicos del sitio web institucional y su interacción con los repositorios. Si la necesidad principal es mostrar datos de repositorio preexistentes con interacción dinámica limitada, un SSG podría ser suficiente. Sin embargo, para escenarios más complejos que involucran filtrado dinámico, contenido personalizado o integración con múltiples plataformas, un CMS sin cabecera podría ser más apropiado.

ANÁLISIS COMPARATIVO: VENTAJAS Y DESVENTAJAS DE LOS CMS ESTÁTICOS Y SIN CABECERA PARA SITIOS WEB INSTITUCIONALES (VS. CMS TRADICIONALES)

Los CMS estáticos y sin cabecera ofrecen varias ventajas significativas en comparación con los CMS tradicionales para sitios web institucionales. En términos de rendimiento, ambos tipos de arquitecturas proporcionan tiempos de carga más rápidos, ya sea mediante páginas pre-construidas en el caso de los SSGs o a través de la eficiente entrega de contenido mediante APIs en los CMS sin cabecera. Esta mejora en la velocidad no solo beneficia la experiencia del usuario, sino que también puede tener un impacto positivo en el posicionamiento SEO. La seguridad es otra ventaja clave, ya que la ausencia de bases de datos y procesamiento del lado del servidor en los SSGs, o la separación de la capa de presentación en los CMS sin cabecera,

reduce la superficie de ataque (Jain, 2021).

La escalabilidad también se ve favorecida en estas arquitecturas modernas. Los archivos estáticos generados por SSGs pueden servirse fácilmente a través de CDNs, mientras que los CMS sin cabecera permiten escalar el *frontend* y el *backend* de forma independiente. Además, los desarrolladores disfrutan de mayor flexibilidad y capacidad de personalización al poder elegir las tecnologías *frontend* que mejor se adapten a sus necesidades y crear experiencias de usuario únicas. En términos de costos, los SSGs pueden ofrecer un alojamiento más económico al poder utilizar plataformas más sencillas o CDNs, y ambos tipos de arquitecturas pueden implicar una reducción en los costos de mantenimiento debido a la menor cantidad de componentes complejos (Diaz, 2018). Los CMS sin cabecera también facilitan un mejor flujo de trabajo para los desarrolladores al permitir que los equipos de *frontend* y *backend* trabajen en paralelo, y promueven la reutilización de contenido a través de múltiples canales (Gustafsson, 2019).

A pesar de estas ventajas, los CMS estáticos y sin cabecera también presentan ciertas desventajas. Su configuración y mantenimiento pueden ser más complejos y requerir habilidades de desarrollo especializadas, especialmente para implementar características dinámicas en SSGs o desarrollar el *frontend* en CMS sin cabecera. Los SSGs carecen de funcionalidades dinámicas integradas, y los CMS sin cabecera no incluyen una capa de presentación, lo que exige la integración de servicios externos o el desarrollo personalizado para características como búsqueda, comentarios y autenticación de usuarios. Los CMS sin cabecera a menudo no ofrecen una función de vista previa de contenido para usuarios no técnicos, y la configuración inicial del entorno de desarrollo y el proceso de construcción en SSGs, o el desarrollo del *frontend* en CMS sin cabecera, puede llevar más tiempo (Diaz, 2018).

El flujo de trabajo de gestión de contenido puede resultar menos intuitivo para usuarios no técnicos en comparación con los CMS tradicionales con editores WYSIWYG, ya que los SSGs suelen utilizar Markdown y los CMS sin cabecera requieren familiaridad con la interfaz de *backend* (Tramullas, 2020). Las actualizaciones o modificaciones de contenido pueden requerir la intervención de desarrolladores, especialmente en CMS sin cabecera donde el *frontend* está separado (Jain, 2021). La implementación de prácticas SEO puede ser más compleja en CMS sin cabecera al no haber un tema integrado, y tanto los SSGs como los CMS sin cabecera generalmente tienen un ecosistema de *plugins* o extensiones más reducido o diferente en comparación con CMS tradicionales como WordPress o Joomla.

La decisión de adoptar CMS estáticos o sin cabecera para sitios web institucionales implica un equilibrio entre rendimiento y flexibilidad, por un lado, y complejidad y facilidad de uso, por el otro. Si bien los beneficios en términos de velocidad y escalabilidad son significativos, las instituciones deben considerar la experiencia técnica requerida y el impacto potencial en los flujos de trabajo de gestión de contenido. Los CMS tradicionales ofrecen una experiencia más integrada y amigable para los creadores de contenido, pero pueden sufrir problemas de rendimiento y vulnerabilidades de seguridad a medida que crecen en complejidad. Los SSGs

y los CMS sin cabecera abordan estos problemas, pero introducen una curva de aprendizaje más pronunciada y podrían requerir un enfoque diferente para la gestión de contenido y las funcionalidades dinámicas.

DECONSTRUCCIÓN DE LA IMPLEMENTACIÓN EXISTENTE: ANÁLISIS DE LOS *PLUGINS* OPENSEARCH Y DSPACE API EN CMS TRADICIONALES BASADOS EN PHP

Para comprender los desafíos de adaptación, es necesario analizar la estructura y el funcionamiento de los *plugins* OpenSearch y DSpace API existentes en los CMS tradicionales. Tomando como ejemplo el *plugin* wp-dspace para WordPress, se puede observar una estructura típica de *plugin* que incluye un archivo principal (ej., wp-dspace.php), archivos de configuración y posiblemente clases personalizadas. Este *plugin* interactúa con la API de WordPress utilizando acciones y filtros para integrar su funcionalidad dentro del entorno del CMS (Villarreal, Manzur, Vila & De Giusti, 2017). Las llamadas a las APIs de OpenSearch y DSpace se realizan utilizando funciones de PHP (Villarreal, Salamone Lacunza, Vila, De Giusti, & Manzur, 2017). Los resultados de la búsqueda se muestran dentro del tema de WordPress, ya sea mediante el uso de *shortcodes* o *widgets*.

En el caso de Joomla, la extensión desarrollada sigue el patrón de arquitectura Modelo-Vista-Controlador (MVC) propio del *framework*, y la interacción con las APIs de OpenSearch y DSpace se realiza desde el servidor utilizando las clases HTTP proporcionadas por Joomla. La presentación de los resultados de búsqueda se logra mediante el uso de layouts y módulos dentro de la plantilla de Joomla.

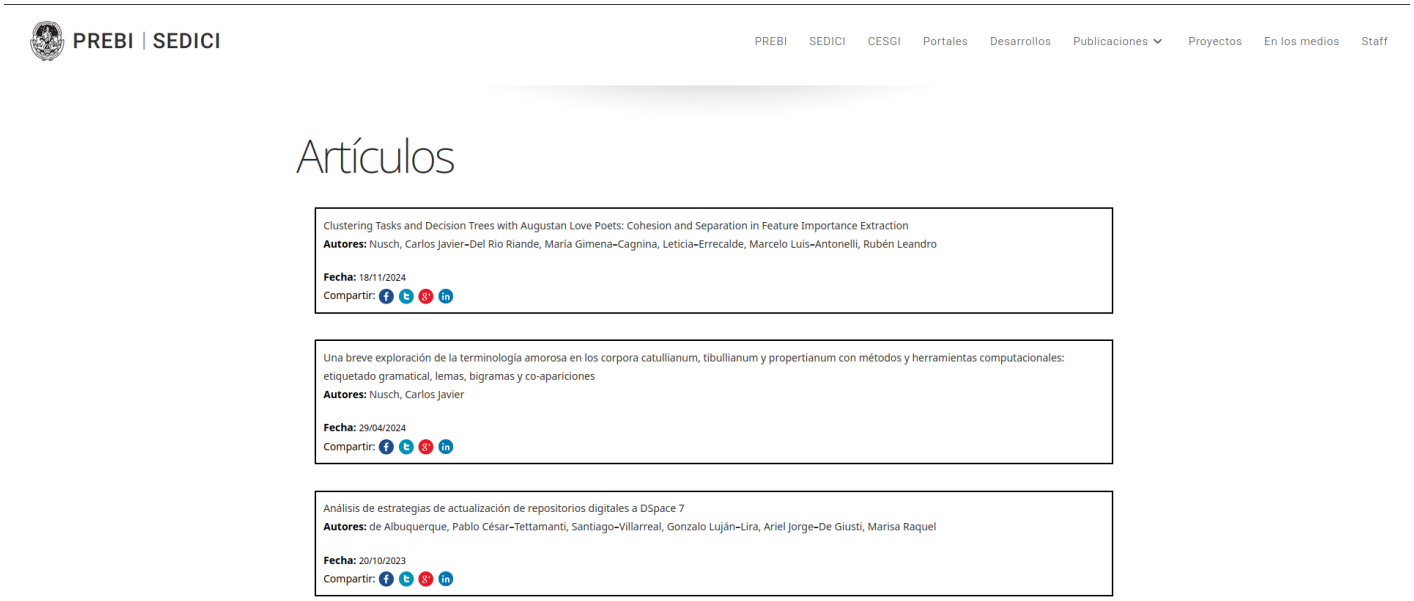
Choique 1.x es un CMS menos conocido, desarrollado por la UNLP hace más de 10 años[1] y utilizado para la implementación del portal principal de la Universidad y de algunos portales institucionales internos. Choique 1.x también fue desarrollado con una arquitectura extensible y sobre PHP, similar a WordPress y Joomla. Versiones posteriores a Choique CMS fueron desarrolladas sobre Ruby y su *framework* Rails, y en la actualidad su desarrollo ha sido descontinuado y su uso es muy acotado.

Los *plugins* existentes están fuertemente acoplados con los ciclos de vida y las estructuras de API específicas de WordPress, Joomla y Choique. Este fuerte acoplamiento será el principal desafío para adaptarlos a la naturaleza desacoplada de los SSGs y los CMS sin cabecera. Los *plugins* de WordPress dependen de acciones y filtros, las extensiones de Joomla siguen el patrón MVC y Choique aprovechaba la arquitectura extensible del *framework symfony.* Estos son mecanismos PHP del lado del servidor que no son directamente aplicables a los entornos del lado del cliente o de tiempo de construcción de los SSGs y los CMS sin cabecera. El código PHP dentro de los *plugins* responsables de interactuar con las APIs de OpenSearch y DSpace podría ser reutilizable, pero el código del *framework* circundante que maneja la integración y el rende-

[1] Desarrollo-CeSPI. (2023). *Choique* [Código de computador]. GitHub. https://github.com/Desarrollo-CeSPI/choique

rizado del CMS probablemente necesitará ser reimplementado o reemplazado con diferentes tecnologías.

Imagen 1: Plugin wp-dspace utilizado en Wordpress para recuperar publicaciones de repositorios



Fuente: print de tela SEDICI (2025)

OBSTÁCULOS TÉCNICOS Y CONCEPTUALES EN LA TRANSICIÓN A ARQUITECTURAS CMS MODERNAS

La adaptación de los componentes de consulta OpenSearch y DSpace API a las arquitecturas de CMS estáticos y sin cabecera presenta varios desafíos técnicos y conceptuales. Uno de los principales obstáculos es la disparidad entre el lenguaje de programación utilizado originalmente (PHP) y el lenguaje predominante en el desarrollo *frontend* para SSGs y CMS sin cabecera, que suele ser JavaScript. Cabe destacar que no se trata sólo de un cambio de lenguaje de programación, sino también de una modalidad de ejecución de código radicalmente diferente: mientras que PHP se ejecuta en el contexto de un servidor web, el código javascript corre típicamente dentro de los navegadores web de los usuarios. Este cambio de paradigma de la ejecución en el servidor a la ejecución en el cliente, o incluso durante el tiempo de construcción, implica repensar cómo se inician las consultas de búsqueda y cómo se muestran los resultados. En el contexto de los sitios estáticos generados por SSGs, no existe un *backend* dinámico que pueda consultar directamente las APIs en tiempo de ejecución. Por lo tanto, la implementación de la funcionalidad de búsqueda requiere estrategias alternativas, como utilizar JavaScript del lado del cliente para llamar a las APIs o pre-construir índices de búsqueda durante el proceso de generación del sitio.

Otro aspecto relevante a considerar es la cantidad de consultas enviadas a los repositorios. En los CMS tradicionales, es posible utilizar mecanismos de caché en el servidor que reducen la necesidad de acceder repetidamente a las fuentes de datos. Sin embargo, en el caso de los CMS sin cabecera, la aplicación *frontend* se encarga de realizar las consultas pero si no dispone de un sistema de caché adecuado para almacenar los resultados obtenidos, deberá realizar estas consultas de manera recurrente. Esto podría provocar una sobrecarga en los repositorios consultados, y atentar contra la filosofía detrás de los CMS estáticos de reducir la carga de trabajo de los servidores, ya que se estará trasladando esta carga a los servidores de los repositorios.

Por otro lado, con respecto al proceso de pre-construcción de sitios usando SSGs deben tenerse en cuenta algunas consideraciones, en especial las relacionadas con la frecuencia de actualización de los datos. Como se ha mencionado anteriormente, en los sitios web institucionales se observa una clara diferenciación entre el contenido que permanece estático durante largos periodos y aquel que presenta actualizaciones frecuentes. Este último tipo de contenido es el que denominamos anteriormente como "contenido dinámico externo" (Tabla 1) y se obtiene mediante consultas a diversas fuentes de datos (Villarreal, Salamone Lacunza, Vila, De Giusti & Manzur, 2017) . En este contexto, resulta conveniente poder reconstruir selectivamente diferentes subconjuntos de contenido con frecuencias de actualización distintas. El objetivo de este enfoque es evitar la necesidad de reconstruir el sitio en su totalidad ante cada cambio, permitiendo así una mayor eficiencia en el proceso de despliegue. Cabe destacar que esta clasificación no se limita a una dicotomía entre contenido estático y dinámico, sino que pueden existir múltiples subconjuntos con distintas frecuencias de actualización. Además, como ya se ha mencionado, el tamaño y la estructura de la institución también influyen en estos tiempos, haciendo aún más relevante la necesidad de un manejo segmentado y eficiente del contenido.

La gestión de la autenticación para las APIs de OpenSearch y DSpace en un entorno del lado del cliente (tanto para SSGs como para CMS sin cabecera) también requiere una consideración cuidadosa para evitar la exposición de credenciales sensibles. Además, la gestión del estado de las consultas de búsqueda y los resultados en el lado del cliente puede ser más compleja que en una aplicación renderizada en el servidor (Beke, 2018).

Las arquitecturas de *plugins* de WordPress, Joomla y Choique son específicas de esas plataformas y no tienen un equivalente directo en los SSGs o CMS sin cabecera. En estos últimos, la funcionalidad a menudo se construye utilizando módulos, componentes o código personalizado dentro del *framework* de *frontend*. Finalmente, los sistemas de templating utilizados en los CMS tradicionales (ej., plantillas PHP) son diferentes de los utilizados en los SSGs (ej., Liquid, Go templates) o *frameworks* de *frontend* (ej., JSX en React, plantillas Vue). La lógica de renderizado de los resultados de búsqueda deberá adaptarse a estos nuevos sistemas.

La transición de un entorno PHP del lado del servidor a un entorno JavaScript del lado

del cliente (o un proceso en tiempo de construcción) para la funcionalidad de búsqueda es el desafío técnico central. Este cambio impacta la forma en que se realizan las llamadas a la API, cómo se maneja la autenticación y cómo se renderizan los resultados. PHP se ejecuta típicamente en el servidor en respuesta a las solicitudes de los usuarios. JavaScript, en el contexto de los SSGs y los CMS sin cabecera, se ejecuta en el navegador del usuario, lo que significa que la lógica para interactuar con las APIs debe moverse del servidor al cliente, y esto tiene implicaciones para la seguridad y el rendimiento. La naturaleza desacoplada de los CMS sin cabecera presenta un obstáculo conceptual para aquellos acostumbrados al enfoque estrechamente integrado de los CMS tradicionales. La separación de la gestión de contenido de la presentación requiere una forma diferente de pensar acerca de cómo se implementan y gestionan muchas características, en particular la búsqueda sobre repositorios institucionales.

ESTRATEGIAS PARA LA IMPLEMENTACIÓN DE FUNCIONALIDADES DE BÚSQUEDA EN CMS ESTÁTICOS Y SIN CABECERA

Para implementar la funcionalidad de búsqueda (OpenSearch y DSpace API) en sitios *web* generados con CMS estáticos (SSGs), se pueden considerar varias estrategias. Una de ellas es la búsqueda del lado del cliente con JavaScript. Esta técnica implica obtener e indexar los datos del repositorio (respuestas de las APIs de OpenSearch y DSpace) durante el proceso de construcción del sitio y almacenarlos como un archivo JSON o JavaScript estático. Luego, se utiliza una biblioteca de búsqueda de JavaScript (como Lunr.js o Fuse.js) en el *frontend* para realizar la búsqueda en este índice pre-construido. Los resultados de la búsqueda se renderizan dinámicamente utilizando JavaScript y el sistema de plantillas del SSG.

Otra estrategia es la integración con servicios de búsqueda externos, como Algolia o Elasticsearch. En este caso, las consultas de búsqueda se envían desde el sitio estático (utilizando JavaScript) a la API del servicio externo, y los resultados devueltos por el servicio se renderizan utilizando JavaScript.

Para escenarios que requieren una lógica del lado del servidor más compleja (por ejemplo, filtrado avanzado, autenticación para el acceso a la API), se pueden utilizar funciones *serverless* (como AWS Lambda o Netlify Functions) que se activan mediante eventos del lado del cliente. La función *serverless* puede entonces interactuar con las API DSpace y devolver los resultados al cliente (Schall, Margaritov, Ustiugov, Sandberg, & Grot, 2022).

En cuanto a los CMS sin cabecera, una estrategia común es el consumo directo de la API en el *frontend*. La aplicación *frontend* (construida con un *framework* como React, Vue.js o Angular) llama directamente a las APIs de OpenSearch y DSpace utilizando bibliotecas HTTP. Es importante tener presente el uso de una caché, como se mencionó en el apartado anterior, para evitar sobrecargar a los repositorios consultados. Asimismo, la autenticación y autorización de la API se gestionan dentro de la aplicación *frontend*, implementando las medidas de seguridad adecuadas. Los resultados de la búsqueda se renderizan dinámicamente dentro del *framework*

de *frontend*.

Otra opción es el patrón *Backend* for *Frontend* (BFF). Esta estrategia introduce una capa de API intermediaria (construida con una tecnología como Node.js o Python) que se sitúa entre el *frontend* y las interfaces de OpenSearch/DSpace. El *frontend* se comunica con el BFF, que se encarga de la agregación de APIs, la lógica compleja y, potencialmente, la autenticación antes de reenviar las solicitudes a las APIs de OpenSearch y DSpace. Esto puede mejorar la seguridad y simplificar el código del *frontend* (Falkevych & Lisniak, 2024).

Finalmente, es importante explorar si el CMS sin cabecera elegido ofrece funcionalidades de búsqueda integradas o permite indexar fuentes de datos externas. Algunos CMS sin cabecera pueden proporcionar estas opciones, lo que simplificaría la implementación de la búsqueda.

La implementación de la búsqueda en SSGs a menudo implica un equilibrio entre la frescura de los datos y la complejidad del proceso de construcción. La pre-construcción de un índice de búsqueda proporciona velocidad, pero requiere reconstruir el sitio cada vez que cambian los datos del repositorio. Las llamadas a la API del lado del cliente ofrecen más datos en tiempo real, pero pueden afectar el rendimiento. Para los repositorios institucionales donde el contenido se actualiza con frecuencia, un índice de búsqueda puramente estático podría no ser ideal. Un enfoque híbrido que utilice llamadas a la API del lado del cliente o que aproveche las funciones *serverless* para la recuperación dinámica de datos podría ser necesario. Los CMS sin cabecera ofrecen más flexibilidad en la implementación de la búsqueda, pero también depositan más responsabilidad en el equipo de desarrollo *frontend*. La elección de la estrategia correcta (consumo directo de la API frente a BFF) depende de la complejidad de los requisitos de búsqueda y del nivel deseado de seguridad y rendimiento. El consumo directo de la API es más simple para funcionalidades de búsqueda básicas. Sin embargo, para escenarios más complejos que involucran múltiples llamadas a la API, agregación de datos o autenticación sensible, una capa BFF puede proporcionar un mejor control y seguridad (Falkevych & Lisniak, 2024).

RENDIMIENTO, SEGURIDAD Y ESCALABILIDAD EN CMS MODERNOS: CONSIDERACIONES PARA LA IMPLEMENTACIÓN DE COMPONENTES DE CONSULTA

Desde el punto de vista de la performance, en el contexto de los SSGs, el rendimiento se relaciona con la optimización del proceso de construcción para manejar grandes conjuntos de datos de los repositorios. Se deben considerar las construcciones incrementales y las técnicas de indexación eficientes. El rendimiento de la búsqueda del lado del cliente depende del tamaño del índice y la eficiencia de la biblioteca de búsqueda de JavaScript utilizada. Para los CMS sin cabecera, el rendimiento del *frontend* está ligado a la eficiencia de las llamadas a la API, las estrategias de almacenamiento en caché y la renderización optimizada de los resultados de búsqueda dentro del *framework* elegido. El uso de CDNs para entregar activos estáticos y el

almacenamiento en caché de las respuestas de la API pueden mejorar significativamente el rendimiento.

En cuanto a la seguridad, en los SSGs se debe evitar la exposición de claves de API o información sensible en el código del lado del cliente. Si se utilizan funciones *serverless*, se debe garantizar su correcta protección (O'Meara & Lennon, 2020). En los CMS sin cabecera, es necesario proteger las claves y *tokens* de la API. Se recomienda utilizar mecanismos de autenticación seguros (como OAuth 2.0) para acceder a las APIs de OpenSearch y DSpace. También se deben tener en cuenta los posibles problemas de Intercambio de Recursos de Origen Cruzado (CORS) al realizar llamadas a la API desde el f*rontend* (Petty & Thompson, 2017). La separación del *backend* en los CMS sin cabecera generalmente mejora la seguridad.

La escalabilidad en los SSGs se gestiona principalmente a través de la plataforma de alojamiento o la CDN que sirve los archivos estáticos (Newson, 2017). El proceso de construcción podría requerir optimización para repositorios muy grandes. En los CMS sin cabecera, el b*ackend* del CMS y la aplicación *frontend* pueden escalar de forma independiente según sus demandas específicas de tráfico y procesamiento. Es importante asegurarse de que las APIs de OpenSearch y DSpace también puedan manejar la carga esperada de la nueva implementación.

La optimización del rendimiento es crucial tanto en los SSGs como en los CMS sin cabecera. Para los SSGs, el tiempo de construcción y la velocidad de búsqueda del lado del cliente son clave. Para los CMS sin cabecera, la eficiencia de las llamadas a la API y el renderizado del *frontend* son primordiales. El almacenamiento en caché puede desempeñar un papel importante en la mejora del rendimiento en ambos escenarios. Un proceso de construcción lento en un SSG puede dificultar las actualizaciones de contenido. Una interfaz de búsqueda lenta en cualquier sistema puede frustrar a los usuarios. La implementación de mecanismos de almacenamiento en caché efectivos en varios niveles (por ejemplo, CDN, navegador, en memoria) puede mejorar significativamente la capacidad de respuesta de la funcionalidad de búsqueda. La seguridad en una arquitectura desacoplada requiere un cambio de enfoque desde la protección del CMS del lado del servidor hacia la protección de la aplicación del lado del cliente y los canales de comunicación con las APIs externas (Gowda & Gowda, 2024). El manejo cuidadoso de las claves de la API y la adopción de prácticas de autenticación seguras son esenciales. La exposición de las claves de la API en el código del lado del cliente puede conducir a un acceso no autorizado. El uso de métodos de autenticación seguros y, potencialmente, un *backend* intermediario para el manejo de operaciones sensibles puede mitigar estos riesgos.

CONCLUSIÓN: NAVEGANDO EL FUTURO DE LA BÚSQUEDA EN REPOSITORIOS INSTITUCIONALES CON TECNOLOGÍAS WEB MODERNAS

En resumen, la adaptación de los componentes de consulta OpenSearch y DSpace API a los generadores de sitios estáticos y los CMS sin cabecera presenta consideraciones y desafíos significativos, principalmente relacionados con la diferencia en las arquitecturas, los lenguajes de programación y los paradigmas de ejecución. Sin embargo, esta transición ofrece beneficios potenciales sustanciales en términos de rendimiento, seguridad y escalabilidad para los sitios web institucionales y sus repositorios.

Comprender las diferencias arquitectónicas fundamentales y los principios básicos de los CMS modernos es esencial para abordar estos desafíos de manera efectiva. Las estrategias de implementación deben considerar el uso de JavaScript del lado del cliente, la integración con servicios de búsqueda externos, el aprovechamiento de funciones *serverless* y el consumo directo de APIs en el *frontend*.

La adopción de mejores prácticas de desarrollo, como el diseño modular, la abstracción de APIs, un enfoque basado en la configuración y la documentación clara, será crucial para crear componentes de consulta que sean fácilmente adaptables a diferentes CMS estáticos y sin cabecera, priorizando su filosofía de rendimiento y flexibilidad.

Mirando hacia el futuro, la búsqueda remota sobre repositorios institucionales continuará evolucionando con el avance de las tecnologías web. La capacidad de integrar eficientemente los sistemas de búsqueda con arquitecturas web modernas será cada vez más importante para garantizar un acceso rápido y seguro a la creciente cantidad de información académica disponible en línea.

REFERENCIAS

Beke, Mathias. (2018). *On the comparison of software quality attributes for client-side and server-side rendering* [Tesis de grado Universiteit Antwerpen, Antwerpen, Belgium]. https://denbeke.be/thesis/versions/mathias-beke-final.pdf

De Giusti, Marisa. R. (2022). *Gestión de la calidad y la preservación en repositorios institucionales*. Actas de la XI Conferencia Internacional de Bibliotecas y Repositorios Digitales (pp. 22–28). Ibero-American Science and Technology Education Consortium.https://www.istec.org/liblink/wp-content/uploads/sites/2/2022/10/Gestion-de-la-calidad-y-la-preservacion-en-repositorios-institucionales.pdf

Diaz, Chris. (2018). Using static site generators for scholarly publications and open educational resources. *Code4Lib Journal, (40).* https://journal.code4lib.org/articles/13861

Falkevych, Vitalii, & Lisniak, Andrii. (2024). Client state management using Backend for Frontend pattern architecture in B2B segment. *Artificial Intelligence, 29(2),* 49–60. https://doi.org/10.15407/jai2024.02.049

Gowda, Priyanka, & Gowda, Ashwath Narayana. (2024). Best practices in REST API design for enhanced scalability and security. *Journal of Artificial Intelligence, Machine Learning and Data Science,* 1(2), 827–830. https://doi.org/10.51219/JAIMLD/priyanka-gowda/202

Gustafsson, Natalie. (2019). *Developing modern web applications with the React library and WordPress Headless CMS.* [Tesis de grado, Laurea University of Applied Sciences, Helsinki, Finland]. Theseus. https://www.theseus.fi/handle/10024/172161

Jain, Vivek. (2021). Headless CMS and the Decoupled Frontend Architecture. *International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, 9*(4), 1–5. https://doi.org/10.5281/zenodo.14752509

Newson, Kaitlin. (2017). Tools and workflows for collaborating on static website projects. *Code4Lib Journal, (38).* https://journal.code4lib.org/articles/12779

O'Meara, Wesley, & Lennon, Ruth. G. (2020). Serverless computing security: Protecting application logic. Serverless computing security: Protecting application logic. *2020 31st Irish Signals and Systems Conference (ISSC)* (pp. 1–5). IEEE. https://doi.org/10.1109/ISSC49989.2020.9180214

Petty, David, & Thompson, Jacob. (2017). *The Not-So-Same-Origin Policy* [Whitepaper]. Independent Security Evaluators. https://www.ise.io/wp-content/uploads/2018/03/ise_same-origin-policy_whitepaper.pdf

Schall, David, Margaritov, Artemiy, Ustiugov, Dimitrii, Sandberg, Aandreas, & Grot, Boris. (2022). Lukewarm serverless functions: Characterization and optimization. *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22)* (pp. 757–770). Association for Computing Machinery. https://doi.org/10.1145/3470496.3527390.

Tramullas, Jesús. (2020). Elaboración de productos de información con JAMstack: del sistema de gestión de contenidos al web estático. *Anuario ThinkEPI*, *14,* pp. 1–7. https://doi.org/10.3145/thinkepi.2020.e14f05

Villarreal, Gonzalo Lujan, Manzur, Ezequiel, Vila, María Marta, & De Giusti, Marisa Raquel. (2017). Interoperabilidad con repositorios digitales: uso de OpenSearch en sitios web institucionales. *Actas de la VII Conferencia Internacional sobre Bibliotecas y Repositorios Digitales de América Latina (BIREDIAL-ISTEC'17) y XII Simposio Internacional de Bibliotecas Digitales (SIBD'17)* (pp. 126–140). Ibero-American Science and Technology Education Consortium. https://sedici.unlp.edu.ar/handle/10915/63566

Villarreal, Gonzalo Lujan, Salamone Lacunza, Paula, Vila, María Marta, De Giusti, Marisa Ra-

quel, & Manzur, Ezequie. (2017). A simple method for exposing repository content on institutional websites. *Open Repositories 2017,* Queensland, Australia. https://sedici.unlp.edu.ar/handle/10915/60507

ANEXO 1

RESUMEN BIOGRÁFICO DE LOS AUTORES

Gonzalo Luján Villarreal

Doctor en Ciencias Informáticas, es director de PREBI-SEDICI de la Universidad Nacional de La Plata, director del Centro de Servicios en Gestión de Información (CESGI, 2016) de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, coordinador informático de revistas científicas de la Universidad Nacional de La Plata y profesor de la Facultad de Informática de la misma universidad. ORCID: 0000-0002-3602-8211

Carlos Javier Nusch

Profesor y Licenciado en Letras por la Universidad Nacional de La Plata y Máster en Humanidades Digitales por la Universidad de Educación a Distancia de España. Ha publicado varios artículos sobre trabajo académico colaborativo, repositorios digitales, digitalización de patrimonio cultural, análisis del discurso político y literatura clásica, medieval y moderna. Trabaja en el Servicio de Difusión de la Creación Intelectual (SEDICI) de la UNLP, en el Proyecto de Enlace de Bibliotecas (PREBI) y en el repositorio CIC-Digital (CICPBA). Es miembro del Comité Asesor del Centro de Servicios en Gestión de Información (CESGI) y personal del Observatorio Medioambiental La Plata (UNLP - CICPBA - CONICET). Coordina la Oficina de Relaciones Institucionales del Consorcio Iberoamericano para la Educación en Ciencia y Tecnología (ISTEC). Participa como docente colaborador ad honorem en el curso de posgrado “Bibliotecas y Repositorios Digitales. Tecnología y aplicaciones” de la Facultad de Informática de la UNLP. Ha participado en proyectos sobre Oralidad, Escritura, Humanidades Digitales Recursos Académicos, Harvesting, OAI-PMH, Visibilidad Web, Repositorios Abiertos, Producción Académica y Científica, Accesibilidad financiados por la UNLP, la CICPBA y el ISTEC. ORCID:0000-0003-1715-4228

Lautaro Josin Saller

Estudiante en Universidad Nacional de La Plata, Provincia de Buenos Aires, Argentina · Programador informático · PREBI-SEDICI. Estudiante de la carrera Licenciatura en Informatica en la Universidad Nacional de La Plata. Experiencia. Programador informático. PREBI-SEDICI.

Juan Cruz Mazullo

Se incorporó al equipo de PREBI-SEDICI en el año 2024. Su labor principal dentro del equipo incluye tareas de diseño y desarrollo de software.