https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 181 - 198

# Automatización de la difusión de la oferta académica mediante un plugin personalizado para WordPress

*Automation of the dissemination of the academic offer through a custom WordPress plugin*

**Información del reporte:**

Licencia Creative Commons

El contenido de los textos es responsabilidad de los autores y no refleja forzosamente el punto de vista de los dictaminadores, o de los miembros del Comité Editorial, o la postura del editor y la editorial de la publicación.

Para citar este reporte técnico:

Cabello Pérez, J. (2026). Automatización de la difusión de la oferta académica mediante un plugin personalizado para WordPress. *Cuadernos Técnicos Universitarios de la DGTIC, 4*(4), páginas (181 - 198). https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196

**Jazmín Cabello Pérez**

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

jazmin.cabello@unam.mx

ORCID: 0009-0006-4338-6494

## Resumen

La administración de la oferta académica de educación continua en la Dirección General de Cómputo y Tecnologías de Información y Comunicación requería que sus plataformas web funcionaran como canales ágiles, precisos y vigentes. Si bien el uso de WordPress ha permitido la descentralización de la publicación de cursos y diplomados, la flexibilidad de esta plataforma resultaba insuficiente para gestionar la difusión en los tiempos disponibles. Al no encontrarse componentes comerciales adecuados para automatizar el proceso de difusión, se optó por desarrollar un componente a la medida.

El desarrollo se centró en la creación de un *plugin* para WordPress que interviniera en el ciclo de vida de las entradas de la oferta académica. La metodología se estructuró en cuatro etapas secuenciales: diseño, configuración, instalación y pruebas. Para la programación del *frontend*, se utilizó JavaScript con el gancho *add_meta_boxes* para incrustar secciones personalizadas en el editor, con capacidad para registrar múltiples combinaciones de horarios y periodos mediante la clonación dinámica de nodos del documento. Durante la configuración de seguridad y persistencia, se implementaron verificaciones de *nonces* y procesos del lado del servidor para almacenar los datos bajo metadatos serializados, aplicando ordenamiento cronológico automático y lógica condicional para alertas de disponibilidad.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 182 - 198

La utilización del *plugin* arrojó indicadores favorables en comparación con la gestión manual. El tiempo de actualización de carteleras académicas se redujo significativamente gracias a la automatización del ordenamiento cronológico, eliminando la necesidad de revisiones diarias. El rendimiento de la plataforma demostró ser altamente eficiente, evitando sobrecargas durante picos de tráfico concurrente. El impacto más relevante se identificó en el flujo de atención al público y conversión de inscripciones, debido a la habilitación de enlaces de correo parametrizados dinámicamente que se activaban cuando la oferta estaba en disponibilidad, estandarizando los mensajes recibidos y optimizando el proceso de registro de aspirantes.

**Palabras clave:** Complementos modulares, gestión institucional automatizada, interfaces dinámicas.

***Abstract***

*The administration of the academic offer of continuing education in the Dirección General de Cómputo y de Tecnologías de Información y Comunicación required its web platforms to function as agile, accurate and up-to-date communication channels. Although the use of WordPress has allowed the decentralization of the publication of courses and diplomas, the flexibility of this platform was insufficient to manage the dissemination within the available timeframe. As no adequate commercial components were found to automate the dissemination process, it was decided to develop a customized component.*

*The development focused on creating a WordPress plugin that intervenes in the life cycle of the academic offer entries. The methodology was structured into four sequential stages: design, configuration, installation, and testing. For frontend programming, JavaScript was used with the add_meta_boxes hook to embed custom sections in the editor, with the ability to record multiple combinations of schedules and periods by dynamically cloning document nodes. During the security and persistence configuration, server-side nonces and process checks were implemented to store the data as serialized metadata, applying automatic chronological ordering and conditional logic for availability alerts.*

*The use of the plugin yielded favorable indicators compared to manual management. The time required to update academic notice boards was significantly reduced thanks to the automation of chronological sorting, eliminating the need for daily reviews. The performance of the platform proved to be highly efficient, avoiding overloads during peaks of concurrent traffic. The most relevant impact was identified in the flow of customer service and conversion of registrations, due to the enabling of dynamically parameterized email links that were activated when the offer was available, standardizing the messages received and optimizing the applicant registration process.*

***Keywords:*** *Modular plugins, automated institutional management, dynamic interfaces.*

# 1. INTRODUCCIÓN

La administración y difusión de la oferta académica de educación continua en la Dirección General de Cómputo y de Tecnologías de Información y Comunicación (DGTIC) requiere que las plataformas web institucionales funcionen como canales de comunicación ágiles, precisos y vigentes. Con el uso generalizado de sistemas de gestión de contenidos (CMS, por sus siglas en inglés) como WordPress, la publicación de cursos y diplomados se ha descentralizado, permitiendo que el personal operativo

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 183 - 198

gestione la información directamente. Sin embargo, la flexibilidad nativa de estas plataformas suele ser insuficiente cuando se enfrenta a las lógicas de negocio y a ventanas de tiempo, críticas en la administración educativa universitaria.

En el caso que aquí se presenta, los responsables de las áreas de pedagogía, relaciones comerciales y mercadotecnia, que participan en el proceso de inscripción, enviaban al departamento de Promoción de Servicios de Docencia en TIC la lista de los cursos a difundir e información adicional relevante para permitir el registro de los interesados. A continuación, la titular del departamento receptor, después de validar la información, solicitaba que ésta se incluyera en la página web destinada para la difusión.

La información recibida se vertía en la sección de productos mediante una plantilla de un tema de WordPress, lo que limitaba severamente las opciones de actualización, ya que cualquier modificación debería hacerse en forma manual y generando código HTML. En vista de que las áreas productoras de la información no cuentan con personal con conocimientos en desarrollo web, el proceso se encontraba limitado severamente por la disponibilidad de tiempo del especialista. Además, el proceso de eliminación de oferta caducada requería una supervisión constante, mientras que el envío de numerosas peticiones de modificación resultaba en ventanas de tiempo de actualización muy reducidas y, a veces, en la exhibición de información caduca. Otro desafío importante lo representaba la gestión de múltiples sedes, calendarios y horarios para un mismo evento de capacitación.

Las soluciones comerciales genéricas disponibles en el ecosistema de *plugins* de WordPress presentaron inconvenientes sustanciales en el contexto institucional, puesto que su rigidez estructural y falta de automatización cronológica resultaba en un incremento de fricción en los canales de atención cuando la oferta poseía cupos limitados. El empleo de herramientas de terceros no integradas derivaba en la visualización de datos obsoletos o en cargas de trabajo redundantes que debía realizar el personal operativo responsable de los sitios web. Esto fue particularmente evidente al valorar los *plugins* The Events Calendar, Sugar Calendar, EventON y Amelia, cuyos desempeños no se adaptaron a los requerimientos específicos de operación. En el Anexo B, se ofrece una breve reseña de las características principales de los cuatro *plugins* antes indicados y de las razones por las que su uso fue considerado inadecuado.

Por lo anterior, se decidió desarrollar una herramienta sencilla y funcional que permitiera descentralizar el registro de los periodos académicos y, al mismo tiempo, dar acceso directo a los responsables de la gestión escolar para administrar sus herramientas sin requerir conocimientos técnicos previos en desarrollo web (Ajimati *et al*., 2025). El componente propio a desarrollar sería un *plugin* para WordPress.

Esta alternativa permite incorporar funcionalidades específicas sin modificar el núcleo de la plataforma, lo que facilita su mantenimiento y evita que las personalizaciones sean afectadas durante las actualizaciones. Este enfoque, recomendado por Pressman y Maxim (2020) para estructurar el *software* mediante componentes con responsabilidades bien definidas, no sólo responde a la defensa y estándares de mitigación de vulnerabilidades web señalados por Likaj *et al*. (2021), sino que coincide plenamente con las directrices de la documentación oficial de WordPress, las cuales promueven el desarrollo y uso de *plugins* para ampliar las funcionalidades de la plataforma bajo criterios rigurosos de organización, seguridad, validación y sanitización del código (WordPress.org, 2023, 2024).

El desarrollo a la medida también posibilita concentrarse únicamente en las funciones que realmente requiere el proceso, mantener un mayor control sobre el código y reducir la dependencia de complementos de terceros. Además, al tratarse de un componente propio, puede mantenerse, actualizarse y ampliarse conforme surjan otras necesidades institucionales.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 184 - 198

En resumen, el objetivo general del proyecto fue desarrollar e implementar un complemento modular y a la medida para WordPress, orientado a automatizar el ordenamiento cronológico de las ventanas de tiempo académicas y optimizar los flujos de comunicación institucional con los aspirantes durante periodos de alta demanda.

# 2. DESARROLLO TÉCNICO

El desarrollo del *plugin* se centró en la creación de un componente nativo para WordPress, con el fin de acelerar la publicación de entradas (*posts*) dedicadas a la oferta académica.

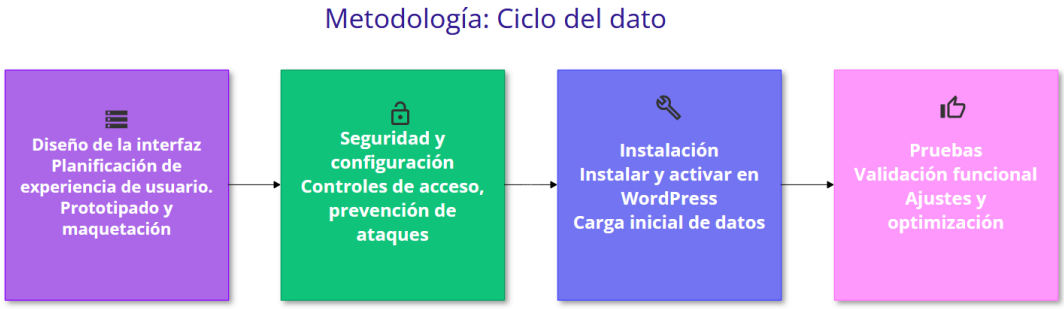
La arquitectura propuesta se orientó a descentralizar la carga de datos sin alterar el núcleo (*core*) del sistema. A través de este desarrollo, se buscó que el personal que realiza la supervisión de la oferta académica dispusiera de controles centralizados e intuitivos en un panel de control del complemento modular, mientras que el usuario final pudiera visualizar una interfaz depurada y con información pertinente, automatizada en tiempo real.

## 2.1 METODOLOGÍA

La metodología se estructuró en cuatro etapas secuenciales, enfocadas en el ciclo del dato (Möller, 2013): diseño de interfaz, configuración de seguridad, instalación y pruebas de despliegue. Las tareas específicas realizadas durante cada una de las etapas se muestran en la Figura 1.

**Figura 1**

*Flujo secuencial y ciclo de datos*




No se trata de una metodología marco genérica (como Scrum o Waterfall), sino de una metodología técnica y procedimental diseñada a la medida para un proyecto de *software* centrado en datos.

En la etapa de diseño de interfaz, se consideraron dos espacios principales: El área administrativa, donde se registran y actualizan los cursos así como su información asociada, y la interfaz pública, donde los usuarios pueden consultar la oferta disponible por fecha y sede.

Para su programación, se utilizó el gancho (*hook*) nativo *add_meta_boxes* para incrustar una sección personalizada en el editor del CMS. Con el fin de resolver la necesidad de registrar múltiples combinaciones de horarios y periodos por curso, se programó una rutina en JavaScript que manipula el Modelo de

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 185 - 198

Objetos del Documento (DOM, por sus siglas en inglés). Al accionar el botón "+ Agregar fecha", el *script* clona dinámicamente un nodo del formulario con campos para las fechas de inicio y fin, días, horarios y una casilla de verificación condicional denominada "Últimos lugares". Asimismo, cada bloque repetible se vincula a un evento (*event listener*) que ejecuta una eliminación manual e inmediata del nodo en el cliente, permitiendo la depuración de datos previa al envío.

En la Figura 2 del Anexo A, se muestra la interfaz desarrollada para que la responsable del departamento capture la información de la oferta académica ya autorizada y sea publicada. Dicha información es la misma que anteriormente se enviaba para su posterior difusión.

La Figura 3 del Anexo A exhibe un ejemplo de la página web en la que aparece publicada la información de sede, fechas y horarios para el evento programado.

Para la etapa de configuración de seguridad y persistencia, el procesamiento de los datos se realizó del lado del servidor mediante la acción *save_post* de WordPress. Como medida de protección frente a solicitudes no autorizadas, se incorporó la verificación de *nonces* (*tokens* aleatorios), mecanismo descrito por Ele (2024) y recomendado como práctica de seguridad para el desarrollo de *plugins* (WordPress, 2026).

Un *nonce* es un valor generado por WordPress que se incorpora a una solicitud para ayudar a comprobar que ésta se originó desde una acción prevista dentro del sitio y permite reducir el riesgo de que determinadas operaciones sean ejecutadas mediante solicitudes externas no autorizadas, particularmente ataques de *Cross-Site Request Forgery* (CSRF).

En el *plugin*, al mostrar el formulario de administración, se genera un campo de seguridad mediante *wp_nonce_field()*. Cuando el usuario guarda o actualiza la información, el servidor recibe nuevamente ese valor y lo verifica mediante *wp_verify_nonce()*. Si la comprobación no es satisfactoria, el procesamiento de los datos se interrumpe y no se ejecuta la actualización correspondiente.

La verificación del *nonce* se complementa con la comprobación de permisos del usuario mediante *current_user_can()* y con la sanitización de los datos recibidos. Esto es importante porque WordPress señala que los *nonces* no deben considerarse mecanismos de autenticación o autorización por sí mismos; deben acompañarse de verificaciones de capacidades para determinar si el usuario tiene permiso para realizar la operación. En el código, esto se materializa precisamente en dos partes:

```
wp_nonce_field(self::NONCE, self::NONCE);
```

Al crear el formulario, WordPress genera el campo de seguridad. Posteriormente, durante el guardado:

```
if (
    !isset($_POST[self::NONCE]) ||
    !wp_verify_nonce(
        sanitize_text_field(wp_unslash($_POST[self::NONCE])),
        self::NONCE
    )
) {
    return;
}
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 186 - 198

Si el *nonce* no existe o no es válido, se ejecuta `return`, por lo que el *plugin* deja de procesar la solicitud antes de modificar los metadatos.

Después, se cuenta con otra protección:

```
if (
    isset($_POST['post_type']) &&
    self::CPT === $_POST['post_type'] &&
    !current_user_can('edit_post', $post_id)
) {
    return;
}
```

Aquí se comprueba que el usuario tenga permisos para editar el registro. Por tanto, es conveniente enfatizar que la protección no depende exclusivamente del *nonce*:

Solicitud → verificación del *nonce* → comprobación de permisos → sanitización → actualización de los datos.

Una vez validada la petición, el sistema procesa el arreglo de datos generado desde la interfaz. Tras comprobar la seguridad del envío, el sistema toma la información del formulario y limpia cada campo de texto para eliminar caracteres o código sospechoso antes de guardarlo.

Finalmente, los bloques eliminados desde la interfaz se excluyen del arreglo y la información resultante se actualiza vía el control automatizado de la visualización de fechas vencidas.

Al tratarse de una estructura de datos organizada como un arreglo, WordPress gestiona su almacenamiento de forma serializada, permitiendo conservar la información asociada a cada entrada bajo una misma clave de metadatos. La serialización consiste en transformar temporalmente una estructura de datos Intel — por ejemplo, un arreglo de PHP— en una representación que pueda almacenarse como un valor dentro de la base de datos. Cuando el *plugin* necesita nuevamente la información, WordPress realiza el proceso inverso y entrega al código el arreglo correspondiente.

En el *plugin* desarrollado, cada emisión se representa conceptualmente de la siguiente manera:

```
$filas = array(array(
   'inicio' => '',
   'fin' => '',
   'dias' => '',
   'horario' => '',
   'ultimos' => '',));
```

A su vez, varias emisiones se integran dentro de un arreglo principal. Una vez que los datos han sido validados y sanitizados, la estructura completa se almacena mediante:

```
update_post_meta(
    $post_id,
    '_ddtic_fh_filas',
    $clean
);
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 187 - 198

No fue necesario programar manualmente la serialización, ya que la API de metadatos de WordPress admite valores serializables y gestiona este proceso durante el almacenamiento y recuperación de los datos. Posteriormente, el *plugin* recupera las filas mediante:

```
$filas = get_post_meta(
    $post_id,
    '_ddtic_fh_filas',
    true);
```

De esta manera, el código vuelve a disponer de la estructura de filas y puede recorrerla para determinar cuáles deben mostrarse en el sitio.

Durante la etapa de instalación y pruebas en el *frontend*, se diseñó código en PHP que recupera el arreglo de metadatos almacenados. Para solucionar la obsolescencia visual de la oferta, el algoritmo aplica un ordenamiento cronológico automático, utilizando funciones de comparación. Éstas contrastan la fecha actual del servidor con el metadato de inicio de cada periodo. De esta manera, se asegura que los cursos vencidos se oculten automáticamente.

Finalmente, el complemento evalúa el estado de la casilla "Últimos lugares". Si esta condición es verdadera, el sistema renderiza un botón de alerta en el cliente y construye dinámicamente un hipervínculo parametrizado bajo el protocolo *mailto*. Las variables de correo institucional, nombre del curso, horarios y sedes se concatenan de forma automática en la cadena de texto. De este modo, al interactuar con el botón, el dispositivo del usuario abre su cliente de correo local con el destinatario, el asunto y el cuerpo del mensaje preconfigurados, lo que elimina posibles errores manuales de transcripción y permite enviar un correo al departamento de Informes y Relaciones asociado al curso para atender la petición de inscripción.

En resumen, la incorporación del *plugin* desarrollado en el proceso de publicación y actualización de la información de los eventos académicos resultó en la redistribución de funciones realizadas. La Tabla 1 muestra la manera en que se ha realizado esta redistribución.

**Tabla 1**

*Relación de funciones de modificación de la oferta académica relacionadas con los responsables de realizarlas*

| Función | Acción de la persona administradora | Acción automatizada por el *plugin* |
|---|---|---|
| Registrar una emisión | Información que captura: inicio, término, días y horario | Almacena los datos mediante las funciones de WordPress |
| Registrar sede | Información que captura: sede | Recupera el dato para mostrarlo en la cartelera |
| Registrar correo | Información que captura: correo de contacto | Lo utiliza para construir el enlace de contacto |
| Vigencia de una emisión | Información que captura: fecha de término | Compara la fecha y deja de mostrar la emisión cuando pierde vigencia |

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 188 - 198

| Función | Acción de la persona administradora | Acción automatizada por el *plugin* |
|---|---|---|
| "Últimos lugares" | Decide la condición y selecciona la casilla en la interfaz | Muestra el botón correspondiente en la página web |
| Correo de "Últimos lugares" | No se programa manualmente el enlace | Genera el `mailto:` con los datos de la emisión |
| Actualizar una emisión | Modifica los campos necesarios | Refleja la información actualizada en la presentación pública |
| Retirar manualmente una fecha vencida | Ya no es necesario como operación habitual | El *plugin* controla su visualización según la fecha de término |

**Uso de inteligencia artificial en el desarrollo tecnológico**

Durante el proceso de diseño y desarrollo del complemento, se emplearon herramientas de inteligencia artificial generativa como apoyo para la generación y optimización de fragmentos de código, la revisión de sintaxis y la consulta de buenas prácticas de programación. Esta decisión se sustenta en evidencia empírica reciente que documenta mejoras significativas en la velocidad de ejecución de tareas de programación cuando se utilizan asistentes de IA como apoyo técnico (Peng *et al*., 2023). Dichas herramientas fueron utilizadas exclusivamente como asistentes técnicos dentro del proceso de desarrollo, sin sustituir el criterio profesional de la persona especialista.

Las decisiones relacionadas con la arquitectura de *software*, el diseño funcional, la integración con WordPress, la validación de requisitos, las pruebas de funcionamiento y la implementación en el entorno institucional fueron realizadas, probadas, valoradas y, en caso de ser necesario, corregidas.

Específicamente, se utilizó Claude 3.7 Sonnet para revisar fragmentos de código PHP, GPT-5 como apoyo en el análisis de los requerimientos funcionales, en la revisión de la estructura inicial del *plugin* y para revisar también fragmentos de código en PHP. Asimismo, se empleó el modelo Gemini 3.1 para el desarrollo armónico de estilos gráficos de los datos publicados en la página web de los cursos.

## 3. RESULTADOS

La integración del *plugin* al flujo de trabajo resultó en un proceso más eficiente en comparación con los esquemas de gestión manuales precedentes. El primer resultado favorable consistió en la reducción del tiempo requerido para la actualización de las carteleras académicas, al permitir modificar la información sin alteración manual del código y mediante la eliminación de las revisiones diarias de vigencia, gracias al ordenamiento cronológico automatizado.

Como referencia, entre enero y agosto 2026, se recibieron 78 solicitudes de actualización, correspondientes a modificaciones de fechas y horarios, retiro de oferta no vigente y colocación de la leyenda "Últimos lugares". De ellas, 58 (74%) fueron resueltas el mismo día de la petición, 15 (19%) al día siguiente, 4 (5%) en un periodo de dos días y una (1%) en un periodo de tres días. Esto se debió a que las actualizaciones dependían no sólo del tiempo efectivo requerido para modificar el código en WordPress, sino también de la disponibilidad de la persona especialista, quien, al tener que realizar otras actividades, no siempre estaba en posibilidad de atender de forma inmediata la petición.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 189 - 198

La actualización manual de la oferta, debido a la necesidad de modificar el código en WordPress, tardaba entre 5 y 20 minutos. Así, el tiempo total destinado a la actualización manual de las 78 solicitudes ascendió a 1,130 minutos (casi 19 horas). En contraste, con el uso del *plugin*, que requiere 4 minutos y 38 segundos (4.63 minutos) para realizar una modificación en los casos más laboriosos, se habría podido lograr en 361.4 minutos (poco más de 6 horas). Adicionalmente, con el empleo del *plugin,* también se habrían ahorrado los 26 días de espera forzosa cuando no se encontraban en posibilidad de atender las peticiones.

En términos de rendimiento de la plataforma web, el uso de arreglos serializados dentro de la arquitectura nativa de metadatos también resultó benéfico. Las consultas a la base de datos relacional no ocasionaron retrasos ni sobrecargas durante los picos de tráfico concurrente, puesto que el sistema evitó la creación de múltiples filas de registros huérfanos, gracias a la depuración inmediata que ejecuta el control de eliminación manual en el panel de edición.

El impacto más relevante se identificó en el flujo de atención al público. La habilitación de los enlaces *mailto:* dinámicos parametrizados bajo la condición de "Últimos lugares" estandarizó por completo los mensajes recibidos en el correo institucional de contacto. Al comparar los históricos de atención, se constató la erradicación de solicitudes con datos ambiguos, incompletos o erróneos en la especificación de horarios y sedes, lo que aceleró la velocidad de respuesta de los centros gestores de la capacitación.

### 3.1 RETOS

El desarrollo de *plugins* a la medida para WordPress, sobre todo cuando se elaboran por primera vez, presenta retos especiales. Para empezar, se requiere que el programador comprenda la idiosincrasia de WordPress, en particular el sistema de *hooks*, con sus distintos *actions* y *filters*. Por ejemplo, decidir sobre si usar *init*, *wp_loaded* o *admin_init* no es obvio para el desarrollador primerizo y sólo se logra la correcta comprensión mediante reiterados intentos y pruebas.

Otro reto al que hubo que enfrentarse fue al conflicto de nombre que se deriva de que todos los *plugins* comparten el mismo espacio global de PHP, lo que, en el presente caso, derivó en modificaciones no intencionadas a los estilos, dado que en algunas ocasiones los nombres de las variables definidas en el tema coincidieron con las empleadas al desarrollar el *plugin*. Para evitar este inconveniente, se optó por usar prefijos únicos.

La seguridad del sitio también representa un reto muy serio al agregar un *plugin* personalizado. La sanitización de las entradas, el uso de *nonces* para proteger los formularios, el escapado de datos de salida y la verificación de permisos no son aspectos opcionales, aunque adquirir estos conocimientos requiere de algunos tropiezos durante el desarrollo.

## 4. CONCLUSIONES

El desarrollo del componente reseñado permitió constatar que la creación de *software* nativo, modular y a la medida representa una solución eficiente y viable para resolver necesidades específicas de comunicación e inscripción en las plataformas web cuando no se tienen alternativas comerciales adecuadas. El acoplamiento correcto de las APIs de WordPress con *scripts* de manipulación dinámica

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 190 - 198

del DOM hizo posible centralizar operaciones complejas de gestión del tiempo en una sola pantalla administrativa de baja complejidad visual.

Se verificó el cumplimiento del objetivo general, demostrando que la automatización del ordenamiento cronológico elimina completamente la exposición de información académica obsoleta en el portal público. De igual manera, la integración de lógica condicional basada en metadatos para activar alertas de disponibilidad demostró ser una estrategia altamente efectiva para mitigar la fricción en la interacción, entendida como el esfuerzo adicional que las personas deben realizar para completar una tarea: esto favorece una experiencia de usuario más eficiente (Norman, 2013; Nielsen, 2024) y facilita un canal directo de comunicación que resguarda la precisión de los datos transmitidos entre el usuario y la institución.

Finalmente, algunas recomendaciones que se consideran pertinentes para encarar los primeros desarrollos de *plugin* para WordPress son: leer el manual oficial de desarrollo de *plugins* de la página web para desarrolladores WordPress.org (2023); aplicar seguridad desde el inicio del desarrollo y no esperar al final del proyecto; revisar código de *plugins* bien hechos para aprender patrones; y comenzar con el desarrollo de *plugins* pequeños.

Para futuros desarrollos en dependencias universitarias con problemáticas similares, se sugiere adoptar esquemas de almacenamiento serializado para proteger el rendimiento de los servidores y documentar los flujos de asistencia con inteligencia artificial bajo principios de transparencia. Este tipo de arquitectura modular no sólo soluciona los retos de la operación cotidiana, sino que establece un estándar escalable apto para integrarse con sistemas institucionales de control escolar mayores.

## REFERENCIAS

Ajimati, M. O., Carroll, N., y Maher, M. (2025). Adoption of low-code and no-code development: A systematic literature review and future research agenda. *Journal of Systems and Software*, 222, 112300. https://doi.org/10.1016/j.jss.2024.112300

Ele, B. I. (2024). An enhanced mechanism for protecting web applications from Cross Site Request Forgery (CSRF). *British Journal of Computer, Networking and Information Technology, 7*(1), 1–17. https://doi.org/10.52589/BJCNIT-R5YYKXKA

Likaj, X., Khodayari, S., y Pellegrino, G. (2021, 6-8 de octubre). *Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks* [ponencia]. RAID '21: 24th International Symposium on Research in Attacks, Intrusions and Defenses, San Sebastián, España. https://doi.org/10.1145/3471621.3471846

Möller, K. (2013). Lifecycle models of data-centric systems and domains. *Semantic Web*, *4*(1), 67–88. https://doi.org/10.3233/sw-2012-0060

Nielsen, J. (2024). *10 usability heuristics for user interface design*. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Norman, D. A. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.

Peng, S., Kalliamvakou, E., Cihon, P., y Demirer, M. (2023). *The impact of AI on developer productivity: Evidence from GitHub Copilot*. arXiv. https://arxiv.org/abs/2302.06590

Pressman, R. S., y Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9.ª ed.). McGraw-Hill.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 191 - 198

WordPress. (2023). *Metadata*. WordPress Developer Resources. https://developer.wordpress.org/plugins/metadata/

WordPress. (2023). *Validating Data*. WordPress Developer Resources. https://developer.wordpress.org/apis/security/data-validation/

WordPress.org. (2023). *Plugin Handbook*. WordPress Developer Resources. https://developer.wordpress.org/plugins/

WordPress.org. (2024). *Detailed Plugin Guidelines*. WordPress Developer Resources. https://developer.wordpress.org/plugins/wordpress-org/detailed-plugin-guidelines/

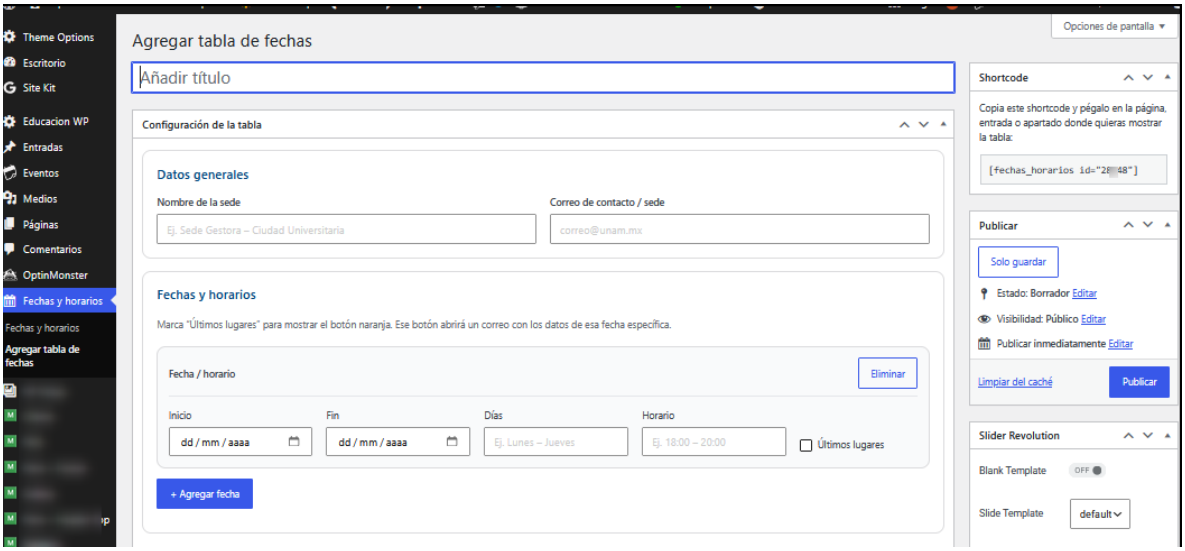
WordPress.org. (2026). *Security*. Plugin Developer Resources. https://developer.wordpress.org/apis/security/

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 192 - 198

# ANEXO A. INTERFAZ DESARROLLADA

**Figura 2**

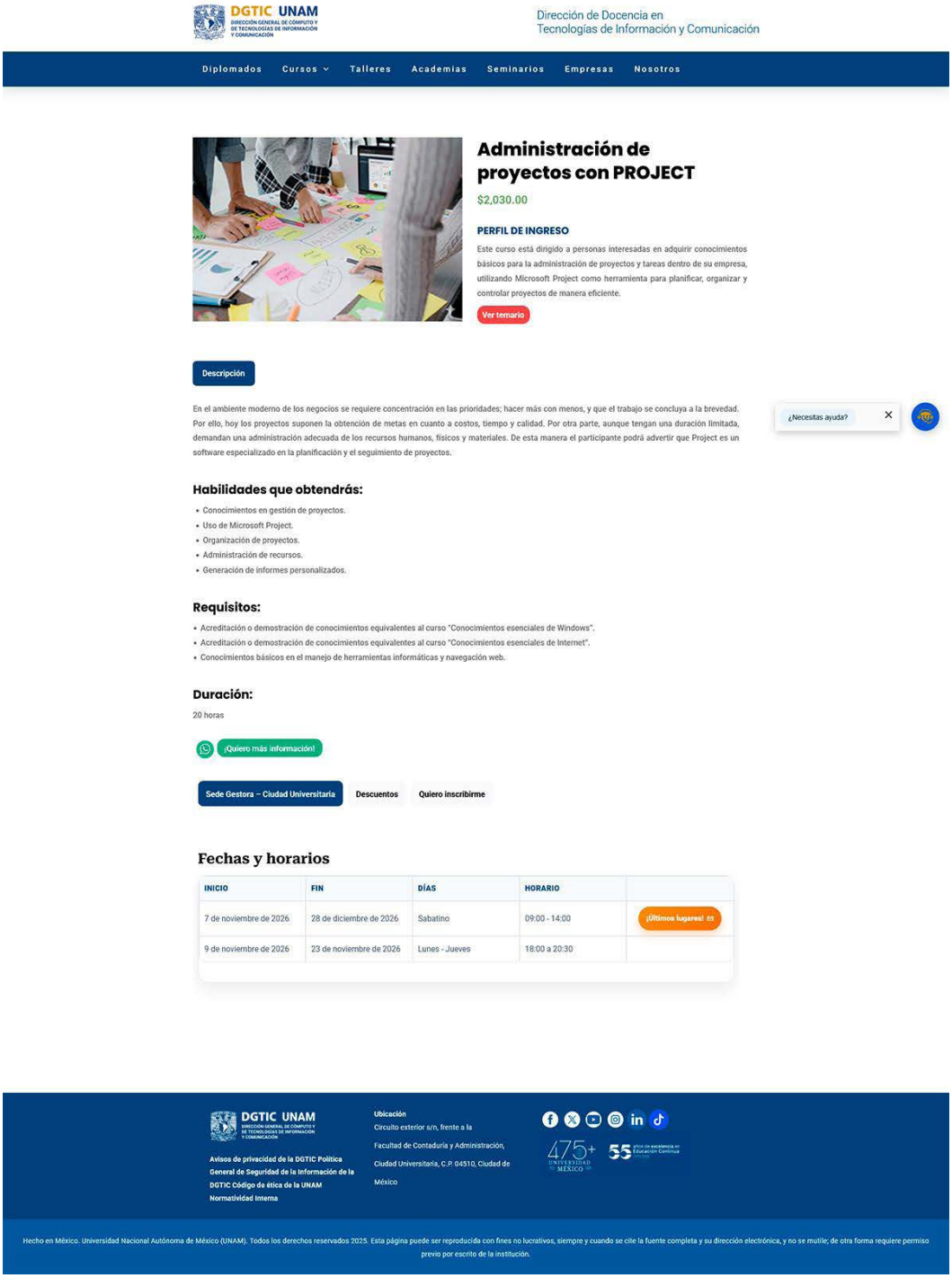
*Interfaz de captura de fechas y horarios en el panel de administración*



https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 193 - 198

**Figura 3**

*Publicación de oferta disponible con la opción de últimos lugares (vista de usuario final)*



https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 194 - 198

# ANEXO B. REVISIÓN DE PLUGINS COMERCIALES PARA LA GESTIÓN DE FECHAS Y EVENTOS EN WORDPRESS

### 1. Propósito del análisis

Antes de desarrollar un *plugin* a la medida para la publicación de fechas y horarios de las actividades académicas, se consideró la posibilidad de utilizar *plugins* existentes para WordPress.

El requerimiento institucional que debería cumplir el *plugin* consistía en incorporar, dentro de páginas web ya existentes, un componente con las siguientes responsabilidades concretas:

1. Registrar varias emisiones de una actividad académica
2. Mostrar fecha de inicio y término
3. Indicar días y horario
4. Identificar la sede gestora
5. Ocultar automáticamente fechas vencidas
6. Señalar determinadas emisiones como “Últimos lugares”
7. Proporcionar un contacto directo con la sede responsable

La revisión tuvo como finalidad determinar si una herramienta disponible comercialmente podía satisfacer los requisitos identificados sin incorporar al sitio web una solución con funcionalidades significativamente mayores a las necesarias.

En el análisis se consideraron cuatro alternativas: The Events Calendar, Sugar Calendar, EventON y Amelia. En una primera revisión documental, se encontró que las principales capacidades y características de los 4 *plugins* giraban alrededor de:

- Administración de eventos y fechas
- Posibilidad de manejar eventos recurrentes o múltiples fechas
- Inserción del contenido dentro de páginas existentes
- Visualización de eventos próximos
- Personalización de la interfaz
- Funcionalidades adicionales incorporadas
- Dependencia de licencias o extensiones
- Adecuación al requerimiento específico del proyecto

Para obtener información objetiva de primera manera, se procedió a instalarlos y probarlos. El objetivo no fue determinar cuál de estos productos es el mejor en términos generales, sino establecer su correspondencia con el requerimiento institucional.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 195 - 198

### 2. The Events Calendar

The Events Calendar es una solución orientada a la administración y publicación de eventos en WordPress. Dispone de herramientas para crear eventos, organizarlos y presentarlos mediante diferentes vistas de calendario. Su ecosistema incorpora funcionalidades avanzadas mediante versiones y extensiones adicionales, entre ellas, eventos recurrentes, filtrado, registro de asistentes y venta de entradas.

Se trata de una alternativa apropiada cuando el sitio requiere que los eventos sean una parte importante de su estructura de información, especialmente cuando se necesita un calendario completo, eventos recurrentes, categorías, filtros, registro de participantes o venta de entradas. Algunas capacidades avanzadas, como la recurrencia, también forman parte del ecosistema comercial del producto.

### 3. Sugar Calendar

Sugar Calendar es un *plugin* especializado en la creación y administración de calendarios y eventos dentro de WordPress. Entre sus características, se encuentran la creación de eventos, calendarios personalizados, eventos recurrentes, etiquetado y filtrado, importación de calendarios, integración con calendarios externos y diferentes formas de presentación.

Además, sus versiones comerciales incorporan características relacionadas con RSVP (peticiones de confirmación), venta de entradas e integraciones. La recurrencia permite configurar patrones diarios, semanales, mensuales o anuales y establecer condiciones más específicas de repetición.

Esta funcionalidad resulta especialmente útil para organizaciones que administran eventos periódicos y necesitan automatizar su generación.

### 4. EventON

EventON es otro *plugin* especializado en calendarios para WordPress. Su propuesta se concentra especialmente en la presentación visual de eventos y ofrece numerosas opciones de configuración. Permite administrar información como fechas, ubicaciones, organizadores y categorías, además de eventos repetitivos. También utiliza *shortcodes* para insertar calendarios y diferentes presentaciones dentro de las páginas del sitio.

Su documentación muestra que el producto dispone de numerosas posibilidades de configuración y puede ampliarse mediante complementos para funciones como RSVP, entradas y otras capacidades relacionadas con la gestión integral de eventos.

Esta flexibilidad representa una ventaja cuando el objetivo es construir un sistema completo de eventos con diferentes vistas y posibilidades de interacción.

### 5. Amelia

Amelia constituye una alternativa con un alcance todavía mayor, ya que está orientado principalmente a la administración de reservaciones, citas y eventos. En el ámbito de eventos, permite manejar calendarios, listas de eventos, reservaciones, asistentes, capacidad, precios, eventos recurrentes y otras funciones relacionadas con la inscripción.

También dispone de *shortcodes* para presentar listas o calendarios de eventos y permite filtrar eventos futuros, pasados o correspondientes a determinados periodos.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 196 - 198

Una característica relevante es que Amelia diferencia entre eventos recurrentes y eventos de varios días. En los primeros, genera ocurrencias independientes; en los segundos, varias fechas forman parte de un mismo evento.

Esta arquitectura resulta útil en escenarios en los que la organización necesita administrar inscripciones, cupos, asistentes, pagos o reservaciones desde WordPress.

## 6. Análisis de los *plugins*

Como se ha visto, existen alternativas consolidadas capaces de administrar eventos en WordPress. Por tanto, la decisión del desarrollo de un *plugin* propio no se tomó con base en la inexistencia de herramientas comerciales, sino en su idoneidad para resolver el problema en el marco de los requerimientos definidos.

**Tabla 2**

*Comparación de capacidades de plugins revisados y los requeridos para el plugin requerido*

| Criterio | The Events Calendar | Sugar Calendar | EventON | Amelia | *Plugin* desarrollado |
|---|---|---|---|---|---|
| Gestión de eventos | Sí | Sí | Sí | Sí | Limitado a fechas de actividades |
| Múltiples fechas / recurrencia | Sí, según funcionalidad | Sí | Sí | Sí | Sí, mediante filas independientes |
| Calendario general | Sí | Sí | Sí | Sí | No requerido |
| Reservaciones | Mediante productos/ extensiones del ecosistema | Según plan/ extensiones | Mediante extensiones | Sí | No |
| Venta de entradas | Disponible dentro del ecosistema | Disponible | Mediante extensiones | Sí | No |
| Gestión de asistentes | Disponible según componentes utilizados | Disponible según plan | Mediante extensiones | Sí | No |
| Inserción mediante *shortcode* | Sí | Dispone de mecanismos de inserción | Sí | Sí | Sí |
| Ocultamiento de fechas vencidas | Configurable según herramienta/ vista | Gestión de eventos y calendarios | Puede trabajar con eventos pasados/ futuros | Permite filtrar eventos futuros/ pasados | Automático y específico |

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 197 - 198

| Criterio | The Events Calendar | Sugar Calendar | EventON | Amelia | Plugin desarrollado |
|---|---|---|---|---|---|
| Botón "Últimos lugares" asociado a correo | Requeriría configuración o adaptación específica | Requeriría adaptación | Requeriría adaptación | Su modelo está orientado a reservas | Incorporado directamente |
| Interfaz específica solicitada | Requiere personalización | Requiere personalización | Requiere personalización | Requiere personalización | Diseñada específicamente |
| Alcance funcional | Amplio | Amplio | Amplio | Muy amplio | Acotado al requerimiento |

Como puede comprobarse al revisar la Tabla 2 de este Anexo, las alternativas estudiadas están orientadas, en distintos grados, hacia la administración integral de eventos: calendarios, recurrencias, vistas especializadas, asistentes, reservaciones, entradas, pagos o integraciones externas. En contraste, el requerimiento institucional es más puntual y, por así decirlo, más modesto.

En el caso del *plugin* The Events Calendar, su incorporación implicaba utilizar una plataforma completa de administración de eventos con una estructura funcional mayor a la requerida, dado que, en la plataforma institucional, los cursos ya cuentan con sus propias páginas y únicamente era necesario incorporar un componente para administrar sus distintas fechas de impartición.

Aunque Sugar Calendar podía cubrir parte de las necesidades, su alcance funcional era mayor al requerido para el componente planteado, debido a que las fechas de los cursos no necesariamente responden a un patrón recurrente. Una misma actividad académica puede impartirse en periodos variables y con horarios diferentes. Además, el propósito no es construir un calendario general de eventos, sino administrar un conjunto reducido de datos asociado a la página de cada actividad.

El empleo de EventON habría requerido configurar o adaptar una solución más amplia para reproducir una interfaz considerablemente más sencilla, ya que era suficiente una tabla integrada visualmente con la página del curso, sin generar una experiencia de calendario independiente.

Finalmente, el *plugin* Amelia no se consideró adecuado puesto que habría sido necesario integrar un sistema de reservaciones completo, con componentes administrativos y de interfaz que no serían aprovechados. Esto obedece a que la inscripción de los participantes no debe ser administrada por el componente desarrollado; únicamente se requería publicar las fechas vigentes y proporcionar un medio de contacto con la sede cuando existieran últimos lugares.

Por lo anterior, se optó por implementar un *plugin* de alcance reducido que utilizara mecanismos nativos de WordPress —*Custom Post Type*, metadatos, *hooks* y *shortcode*— y cuya interfaz pudiera ajustarse directamente a la estructura visual existente.

Se reitera que esta decisión no implica que los *plugins* comerciales analizados sean ineficientes. Por el contrario, pueden resultar más convenientes cuando los requerimientos incluyan administración integral de eventos, recurrencias complejas, venta de entradas, registro de asistentes, reservaciones, pagos, sincronización con calendarios externos o gestión de eventos a mayor escala.

https://doi.org/10.22201/dgtic.30618096e.2026.4.4.196 Vol. 4, Núm. 4. octubre-diciembre 2026, págs. 198 - 198

## 7. Conclusión

La exploración de cuatro *plugins* alternativos permitió ampliar los elementos considerados antes de optar por el desarrollo a la medida.

The Events Calendar, Sugar Calendar, EventON y Amelia ofrecen soluciones maduras para diferentes escenarios de administración de eventos. Sin embargo, el problema identificado requería una funcionalidad más acotada y estrechamente integrada con las páginas existentes de cursos.