https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111

Vol. 3, Núm. 2. abril-junio 2025, págs. 54 - 80

Desarrollo de aplicaciones a la medida con apoyo de modelos GPT

Información del reporte:

Para citar este reporte técnico:

Ramírez Romero, L. M. (2025). Desarrollo de aplicaciones a la medida con apoyo de modelos GPT. *Cuadernos Técnicos Universitarios de la DGTIC*, 3 (2)(54 - 80).

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111

Luz María Ramírez Romero

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

luzrr@unam.mx

ORCID: 0000-0002-8867-9374

Resumen

Con base en la necesidad de mejorar la productividad en el desarrollo de aplicaciones a la medida, en la Dirección General de Cómputo y Tecnologías de Información y Comunicación se exploraron diferentes metodologías, técnicas y herramientas. Se revisó que los modelos GPT inciden en la mejora del aprendizaje de los *frameworks* y lenguajes de programación de los desarrolladores de aplicaciones a la medida, en aras de reducir el tiempo de desarrollo de las diferentes piezas de software que conforman dichas aplicaciones. En este reporte, se explican los elementos que se pueden integrar en el diseño del *prompt* o solicitud y cómo se puede refinar el *prompt* para que los modelos GPT brinden mejores respuestas, más fáciles de integrar a la aplicación en desarrollo y menos complejas en su lógica. Se tomó en cuenta la complejidad de los componentes con la herramienta *laravel sail insights*. Se concluyó que al aplicar los modelos GPT de Inteligencia Artificial generativa se reduce el tiempo de construcción de funciones, componentes y piezas de software, así como el número de incidencias por resolver; además, las incidencias detectadas por el equipo de aseguramiento de la calidad del software se solucionan en menor tiempo. El desarrollador con experiencia puede diseñar los mejores *prompts* para obtener las mejores respuestas de los modelos GPT, y analizar los resultados para decidir la incorporación del código generado, además de tomar en cuenta el marco de ética de los derechos de autor.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 55 - 80

Palabras clave:

Productividad, desarrollo de aplicaciones, ingeniería de prompts, modelos GPT, ingeniería de software

Abstract

Based on the need to improve productivity in custom application development, experts at Dirección General de Cómputo y de Tecnologías de Información y Comunicación explored different methodologies, techniques and tools. It was reviewed that GPT models contribute to improve the learning curve of custom application developers in programming frameworks and languages, thereby reducing the development time of the various software components that make up custom applications. This report explains the elements that can be integrated into the design of the prompt or request and how the prompt can be refined so that GPT models provide better responses that are easier to integrate into the application under development and less complex in their logic. The complexity of the components was taken into account using the Laravel Sail Insights tool. It was concluded that applying generative Artificial Intelligence GPT models reduces the construction time of functions, components and software components, as well as the number of issues to be solved. Furthermore, issues detected by the software quality assurance team are dealt with in less time. An experienced developer can design the best prompts to elicit the best responses from GPT models and analyze the results to decide whether to incorporate the generated code, while also taking into account the ethical framework of copyright issues.

Keywords:

Productivity, application development, prompt engineering, GPT models, software engineering.

1. INTRODUCCIÓN

Las aplicaciones a la medida son algunas de las herramientas que utilizan las organizaciones para acercarse al logro de sus objetivos. El proceso de desarrollo de dichas aplicaciones requiere la inversión de recursos humanos y financieros, así como el tiempo.

En la Dirección General de Cómputo y Tecnologías de Información y Comunicación (DGTIC), se construyen aplicaciones para apoyar las actividades académico-administrativas de entidades y dependencias universitarias que lo solicitan a la Dirección General; es importante revisar e integrar métodos, técnicas y herramientas que mejoren la eficiencia en el desarrollo de esta tarea. Por lo tanto, una de las líneas de análisis es el uso de los modelos de inteligencia artificial generativa para este propósito, en los ámbitos de: reducción de incidencias, disminución del tiempo de desarrollo y del tiempo de corrección de incidencias.

El objetivo de este reporte técnico es compartir cómo se ha utilizado el modelo GPT-4 de inteligencia artificial generativa a través del diseño de *prompts* (solicitudes o instrucciones para los modelos GPT), a fin de disminuir el tiempo de desarrollo de software y reducir las incidencias en los sistemas de información, así como acortar el tiempo de corrección de defectos. En específico, se utilizó el diseño de *prompts* en el

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 56 - 80

Sistema de Censo de Tecnologías de Información y Comunicaciones, durante el periodo de abril de 2024 a febrero de 2025.

2. ANTECEDENTES

El concepto de Inteligencia Artificial (IA) surge en 1950, con el artículo escrito por Alan Turing denominado *Computing Machinery and Intelligence* (Computadoras e inteligencia), en el que propuso que el procesamiento de las computadoras fuera tan avanzado y tan similar a la forma de procesamiento del cerebro humano, que se pudieran confundir las respuestas de la computadora con las que daría una persona. A dicho planteamiento, se le conoce como la *Prueba de Turing* (Robisco, 2024).

La IA tuvo varios desarrollos y aportes, sin embargo, este análisis se centrará en el modelo denominado GPT, desarrollado en la empresa *OpenAI*, que se trabajó a partir de 2018. El modelo GPT ha sido diseñado para generar textos, imágenes, traducciones lingüísticas, conversaciones, resúmenes y código de programación a partir de incorporar técnicas como el procesamiento de lenguaje natural (PLN) y de *Machine Learning* (aprendizaje automático), entre otras (Brown et al., 2020).

El procesamiento de lenguaje natural permite introducir en el modelo un *prompt*, solicitud o instrucción en lenguaje natural y que el modelo pueda transformarlo en *tokens* o representaciones numéricas que pueda interpretar (Martí et al., 2002).

La técnica *Machine Learning* habilita al modelo para recibir o buscar en grandes fuentes de datos (*Big Data*[1]), o en diferentes fuentes o sitios especializados en Internet, y detectar patrones y/o reglas que le permitirán "aprender", de tal forma que el modelo puede generar respuestas sin haber sido programado específicamente para ello, basado en los patrones que reconoció (Ramana et al., 2024).

Además, se consultó el estudio *Stackoverflow Developer Survey* 2024, como referencia para saber qué tanto se están utilizando los modelos de IA GPT en el desarrollo de software a la medida. La encuesta indica que el 76% de desarrolladores a nivel mundial ya están utilizando la IA, o planean hacerlo, con aplicaciones en el desarrollo de aplicaciones a la medida (*2024 Stack Overflow Developer Survey*, s/f).

Otra estadística que muestra la encuesta es que el 82.1% de los desarrolladores están usando ChatGPT, seguidos por el 41.2% de programadores que están utilizando Microsoft Copilot.

2.1 METODOLOGÍA

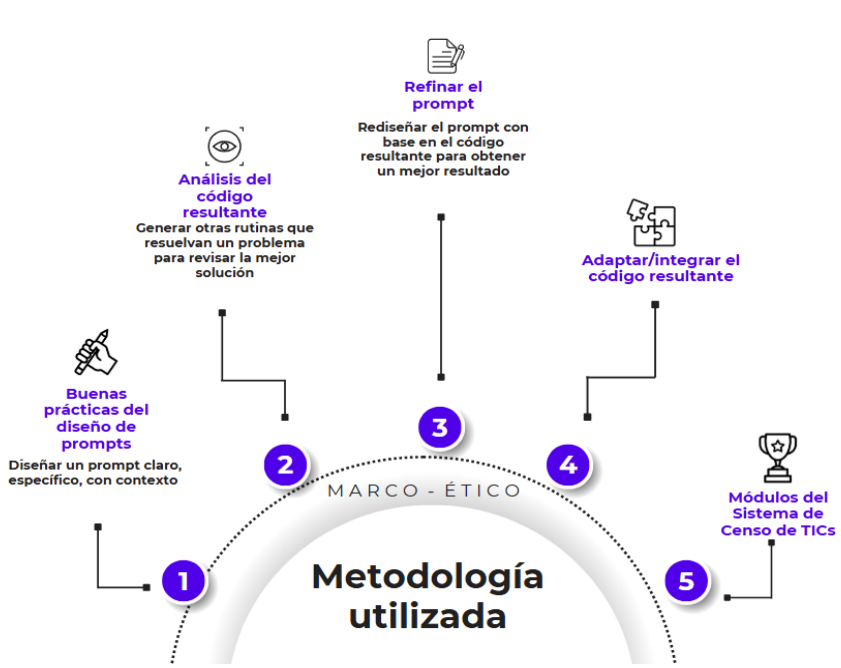
Se revisaron recomendaciones y buenas prácticas de ingeniería y diseño de *prompts*, mismas que se probaron y se aplicaron en el desarrollo del *Sistema de Censo de Tecnologías de Información y Comunicaciones* en la DGTIC. A partir de los resultados obtenidos, se analizaron los *prompts* que producían el código más preciso. También se fue refinando el código a partir de la mejora de los *prompts*, además de que se cuidó el aspecto ético en este proceso. Ver Figura 1.

1 *Big Data*: Término que se utiliza para referirse a grandes volúmenes de datos generados cada segundo, que deben almacenarse y procesarse con técnicas diferentes a las tradicionales como las bases de datos SQL, ya que su procesamiento debe ser en tiempo real; además, los datos pueden estar estructurados, no-estructurados o puede haber combinación de ambos.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 57 - 80

Figura 1

Representación gráfica de la metodología utilizada



Además de efectuarse la corrección de incidencias de los componentes desarrollados, el trabajo de diseño de *prompts* se aplicó en los siguientes componentes para el desarrollo del sistema:

- Registro, edición y borrado de equipos periféricos de digitalización e impresión.
- Consulta tabular, registro, edición y eliminación de equipos de red con contrato de mantenimiento, equipos de telefonía, multilíneas/conmutadores, fibra óptica, puntos Ethernet y servicios de red, así como la integración de nuevas categorías de telefonía y Ethernet.
- Registro y edición de la cantidad de equipo de cómputo con mantenimiento preventivo y correctivo, en la categoría de equipos de cómputo.
- Componentes para registro de la terminación del censo, con la generación del acuse de recibo y la notificación al usuario correspondiente (Responsable TIC).

Las recomendaciones y prácticas se probaron para el desarrollo de código sobre la base tecnológica de los *frameworks* Laravel 11, Livewire 3 y PHP Orientado a Objetos versión 9 y para la construcción de sentencias SQL para el manejador de base de datos PostgreSQL, que es la tecnología que se utilizó en la implementación del sistema en mención.

Se emplearon dos modelos de IA: Microsoft Copilot y ChatGPT, ambos basados en el modelo GPT-4. Se seleccionaron porque han sido entrenados utilizando una gran base mundial de código fuente (GitHub), tomando en cuenta sólo el código abierto, respetando las licencias y derechos de autor de los programadores (iac, 2024).

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 58 - 80

2.2 APLICACIÓN DEL MODELO GPT-4 DE IA EN DIFERENTES ASPECTOS DEL DESARROLLO DE SOFTWARE A LA MEDIDA

Se aplicó el modelo de IA en el desarrollo del *Sistema de Censo de Tecnologías de Información y Comunicaciones*, durante el periodo de abril de 2024 a febrero de 2025. Se obtuvieron reducciones en el número de incidencias (ver Tabla 1), así como mejoras tanto en los tiempos de desarrollo de componentes como en la corrección de incidencias del código.

Tabla 1

Incidencias registradas en sistemas desarrollados del 2023 al 2024[2]

Sistema	Incidencias con causa-raíz: desarrollo-código
Censo * (Usando *prompts* y modelo GPT)	44
Sistema 2 (Cor)	55
Sistema 3 (JG)	54

El modelo GPT-4 fue aplicado en las siguientes áreas del desarrollo del sistema en mención:

1. Corrección de errores: Al desarrollar aplicaciones, los programadores pueden introducir defectos en el código. Los más sencillos de corregir por el modelo de IA GPT-4 son los errores de sintaxis, basta con copiar la sentencia que ha generado el error y obtendremos el código modificado. Si brindamos más contexto al modelo de IA, la respuesta para corregir el error será más acertada. Ver ejemplo en el Anexo 1.

2. Generación de código para atender rutinas comunes: Resulta conveniente contar con un ayudante tecnológico que genere funciones que atiendan problemáticas comunes; así, la mente del programador estará más ocupada en resolver problemas más complejos. Ver ejemplo en el Anexo 2.

3. Generación de consultas SQL (*Structured Query Language*): Los modelos de datos de sistemas reales comúnmente conllevan una fuerte complejidad para obtener cierta información que implique relacionar varias entidades, agrupar varios campos, filtrar la información a partir de algún resultado parcial (*subqueries*) y procesar los datos para dar algún formato al resultado de la consulta. El modelo de IA es de gran apoyo para generar sentencias SQL complejas. Ver ejemplo en el Anexo 3.

4. Búsqueda de alternativas de código: En el ámbito de desarrollo de aplicaciones, siempre es importante llegar a la implementación idónea en eficiencia del uso de recursos de cómputo y de red. Por lo anterior, se puede interrogar al modelo de IA en la generación de código alternativo que encuentre vías más eficientes que las implementadas en el sistema, dentro de las cuales se detecte que se está sobrecargando algún recurso de cómputo o red; esto se puede lograr con herramientas de depuración como *Debugbar de Laravel* o la exploración de los elementos enviados a la red en la *consola del navegador*, o bien utilizando comandos como *sail artisan insights* en *Laravel Sail*. Ver

2 Los datos fueron tomados de la herramienta Mantis Bug Tracker, que sirve para la gestión de incidencias y errores de proyectos de desarrollo de software.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 59 - 80

ejemplos en el Anexo 4 y Anexo 5.

5. Generación de código para *front-end:* Aunque ha resultado un poco limitado en cuanto al control de posiciones fijas y/o relativas de los diferentes elementos en la interfaz de usuario, es posible apoyarse en los modelos de IA GPT-4 para generar elementos complejos en los archivos .blade.php que alojan la interfaz de usuario, como la construcción de vistas y datos agrupados en tablas. Ver ejemplo en el Anexo 6.

6. Búsqueda de librerías de código: Muchas rutinas o procedimientos recurrentes en los sistemas de información ya tienen su solución implementada y compartida por otros programadores en librerías de código disponibles en Internet. Se puede interrogar al modelo GPT-4 para que sugiera las librerías más actuales, o que cuenten con soporte, para integrar al sistema y resolver de forma rápida ciertos problemas comunes y repetitivos. Ver ejemplo en el Anexo 7.

7. Acortar la curva de aprendizaje: Resulta muy ágil interrogar al modelo GPT-4 en la búsqueda de documentación, presentación de ejemplos de uso de determinadas funciones, sentencias o componentes. Asimismo, se puede copiar alguna función y pedirle al modelo de IA que brinde la explicación desglosada de dicha función. Ver ejemplo en el Anexo 8.

La Figura 2 resume los diferentes productos o resultados de los modelos de IA para dar soporte al desarrollo de aplicaciones a la medida.

Figura 2

Productos obtenidos de modelos IA para el desarrollo de aplicaciones a la medida

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 60 - 80

2.3 ANÁLISIS DE PRÁCTICAS PARA EL DISEÑO DE PROMPTS PARA HACER MÁS EFICIENTE LA FASE DE DESARROLLO DE SOFTWARE A LA MEDIDA

El diseño de *prompts* requiere conocimientos técnicos de quien los desarrolla; si se sabe hacer la solicitud al modelo, la respuesta será de mayor ayuda y, posiblemente, el código resultante requiera menos adaptaciones para su integración al sistema (Solohubov et al., s/f).

- Se requieren conocimientos técnicos para poder desarrollar *prompts* que generen código más acertado y correcto. Es mejor brindar estructura técnica al *prompt* para obtener una respuesta que resuelva mejor el problema.
- Se debe conocer el requerimiento del componente, función o rutina para poder expresar la tarea a realizar por el código. Al igual que en la redacción de un documento, se deben evitar las repeticiones innecesarias o pleonasmos que dificulten al modelo la interpretación de la instrucción.
- Usar lenguaje sencillo en la redacción del *prompt* para evitar agregar complejidad en el procesamiento de la instrucción.
- Utilizar verbos más descriptivos o claros.
- *Prompts,* tareas a realizar o código a obtener por el modelo de GPT-4 que incluyan detalles muy concretos y específicos del resultado que se desea obtener. Por ejemplo, se puede dar el nombre de las tablas y el nombre de los campos para generar el *query* de *joins* complejos y se puede pedir al modelo de IA que el *query* brinde formato a alguno o algunos de los campos del resultado (Jarrell, s/f).
- Refinación del *prompt* - Cuando el código generado por la IA se aprecia complejo, se puede mejorar el *prompt* agregando más contexto como, por ejemplo, indicarle al modelo que se desea que el código generado cumpla con un patrón de desarrollo como el *Return early pattern*[3] (Dev, 2020).
- Si los resultados obtenidos después de varias iteraciones en la mejora del *prompt* no son los deseados, se puede probar con otro modelo GPT-4. En el caso práctico del sistema en desarrollo, se usó inicialmente Microsoft Copilot, pero algunas respuestas no daban solución a lo solicitado en el *prompt* y se interrogó a ChatGPT. Se observó que, en ocasiones, Copilot brindaba la mejor respuesta, pero, a veces, la mejor respuesta la generaba ChatGPT, lo que podría abordarse en un estudio posterior.
- Se interrogó tanto a ChatGPT como a Microsoft Copilot si se obtienen mejores respuestas preguntando en algún idioma. La respuesta obtenida fue que es indistinto el idioma en el que se interrogue al modelo. En la refinación de la pregunta, el nuevo *prompt* fue si la mayor parte del entrenamiento del modelo estaba en inglés, ambos modelos respondieron que sí es posible que la mayor parte de los datos de entrenamiento estén en inglés.
- Cuando el código que sugiere el modelo GPT-4 marca errores de ejecución, es posible hacer un nuevo *prompt* incluyendo el mensaje de error obtenido, lo cual brinda al modelo mayor contexto. En la mayoría de los casos, la corrección que brindó el modelo funcionó correctamente y solucionó el error de ejecución.

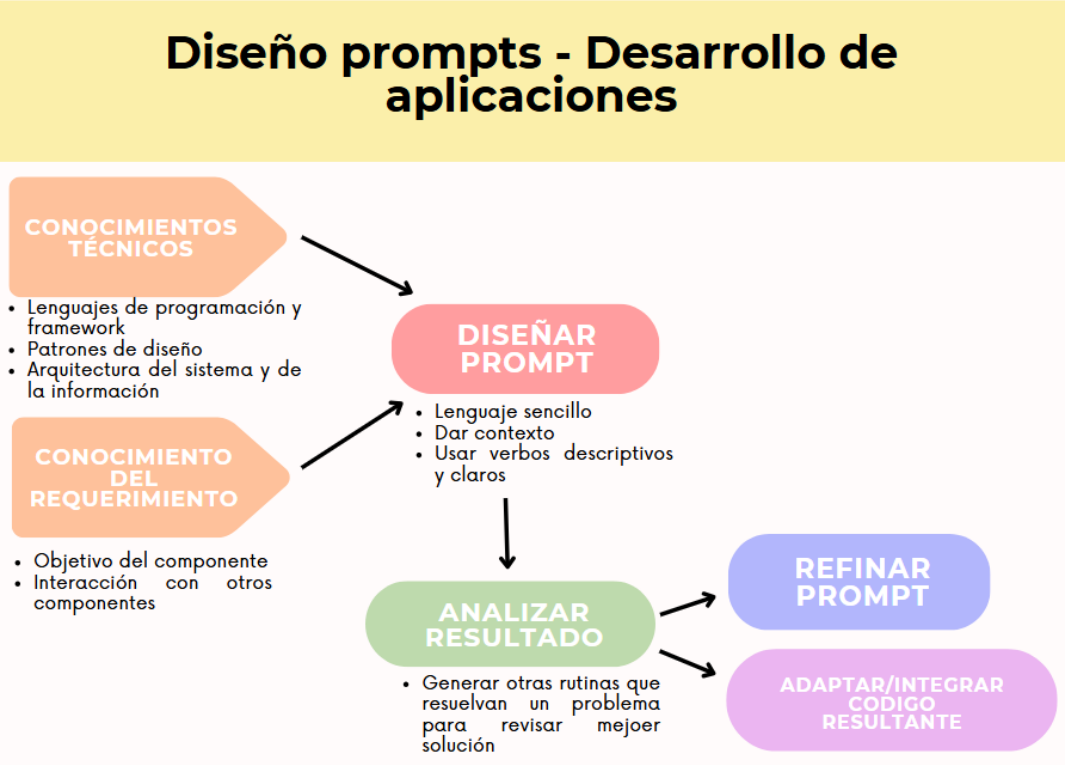
La Figura 3 esquematiza la aplicación de prácticas para el diseño de *prompts*.

3 Return early pattern - técnica de programación que mejora la legibilidad y la mantenibilidad del código al manejar casos especiales o errores lo antes posible en una función o método.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 61 - 80

Figura 3

Diseño de prompts para el desarrollo de aplicaciones



2.4 ÉTICA EN EL DISEÑO DE PROMPTS PARA EL DESARROLLO DE SOFTWARE

Se debe revisar el aspecto ético al utilizar cualquier modelo de IA generativa, en el sentido de no incluir código protegido por derechos de autor, mismo que se puede identificar porque incluye muchos comentarios detallados de lo que está realizando el código, referencias a fuentes externas, así como explicación detallada de variables y estructuras de datos.

En su mayoría, Copilot tiene como fuente de aprendizaje el código abierto de GitHub y patrones y ejemplos abiertos en la red (iac, 2024). En el caso de ChatGPT, sus fuentes de aprendizaje son sitios y repositorios públicos en Internet, por lo cual, no tiene acceso a repositorios privados.

Sin embargo, si el código parece muy complejo y muy específico, es muy probable que haya sido desarrollado por un programador humano (*Developers Warned*, s/f).

Para evitar caer en la utilización de código fuente protegido por derechos de autor, una práctica sugerida es dividir el problema en unidades funcionales más simples y diseñar los *prompts* para obtener rutinas de estas unidades más simples y más genéricas. Estas rutinas genéricas serían integradas con la habilidad de programación del humano para resolver el problema original. De esta manera, el desarrollador está obteniendo apoyo de la herramienta sólo en implementar funciones comunes y ordinarias, siendo así su intelecto el que está resolviendo el problema.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 62 - 80

Unidades de funcionalidad más específicas que parezcan protegidas por derechos de autor deben de ser programadas por el humano, a fin de respetar la originalidad y la autoría, aunque el humano puede también aprender de estas rutinas. Asimismo, resulta importante detallar el código que se desea obtener con el modelo GPT-4, ya que, si no se especifica lo suficiente, el modelo puede generar prácticas que impliquen sesgos indeseables, aprendidos por el modelo a partir de otros sistemas. (Degli-Esposti, 2023).

Otro aspecto ético a cuidar es analizar la sensibilidad de los procesos a automatizar. Por ejemplo, sería irresponsable interrogar al modelo GPT-4 para que genere rutinas relacionadas con la verificación de la identidad de un usuario, ya que el modelo aprende y podría utilizar el *prompt* y el resultado para resolver algún problema de la misma temática en otra organización. Tampoco deben incluirse datos sensibles en el diseño del *prompt*.

Por ello, resulta una buena práctica no incluir la información, ni los procedimientos sensibles, o bien, instalar algún modelo GPT de manera local, a fin de que estos no queden expuestos en algún momento, o bien, desarrollar y construir, sin apoyo de modelos GPT, este tipo de automatizaciones. (Javier Garzás en LinkedIn, s.f).

A continuación, se enlistan algunas aplicaciones de IA generativa que son gratuitas, de código abierto y que pueden instalarse localmente y que no necesitan de conexión a Internet para generar código:

- Llama (*Large Language Model Meta AI*)
- Qwen de Alibaba Cloud
- Gemma de Google AI
- Ollama de la empresa homónima
- Phi de Microsoft

3. RESULTADOS

Se observó que la utilización de modelos GPT y el diseño de *prompts* contextualizados y breves ayudaron a reducir el número de incidencias encontradas atribuibles a defectos en el proceso de desarrollo de código, como se mostró en la Tabla 1.

En seguimiento a los ámbitos mencionados al inicio de este reporte técnico, se puede comentar que:

1. Reducción de incidencias

La Tabla 1 muestra el número de incidencias registradas en tres sistemas desarrollados, cuya causa-raíz es directamente el desarrollo de componentes. Usando el modelo GPT, se observa una disminución en el número de incidencias cuya causa-raíz es desarrollo-código.

2. Disminución del tiempo de desarrollo de software

   Se observó una disminución de un 20 - 30% en el tiempo de desarrollo de componentes con el uso de modelos GPT[4]. Cabe mencionar que en esta estadística personal se ha considerado el tiempo

4 Estimaciones basadas en *Personal Software Process* (PSP), en donde cada desarrollador registra sus estadísticas

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 63 - 80

que conlleva diseñar el *prompt*, analizar el código generado e integrar el código al sistema.

3. Reducción del tiempo de corrección de incidencias

   Se registró una disminución de un 20 - 40% en el tiempo de corrección de defectos con el uso de modelos GPT[5].

Se observó una disminución en el tiempo de desarrollo de componentes de un 20 - 30%, si se considera la inversión de tiempo en el diseño del *prompt,* en su refinamiento a través de varias iteraciones para su mejora, en el análisis del código generado, en la revisión de la eficiencia de éste y en la adaptación para la integración.

A pesar de esta inversión de tiempo, se obtienen resultados eficientes, ya que los modelos GPT se basan en el código desarrollado por expertos humanos, con lo cual, el código generado puede contener funciones que el programador humano desconozca y que acorten o hagan más eficiente la automatización de algún proceso.

También hubo una disminución del tiempo para la corrección de incidencias entre un 20 - 40%. En ocasiones, no tiene tanto impacto el uso de los modelos GPT en la corrección de incidencias, ya que son demasiado específicas y requieren mayor trabajo por parte del desarrollador; sin embargo, en los mejores casos, se disminuye hasta en un 40% de tiempo, ya que las respuestas del modelo son inmediatas y no se requiere leer tanta documentación para poder corregir el código.

Se observó un ahorro de tiempo al obtener resultados inmediatos en la generación de código por los modelos GPT, ya que se basan en soluciones ya implementadas que van incluyendo en su modelo de entrenamiento.

Hubo mejora en la calidad del código, puesto que los componentes generados toman en cuenta las mejores prácticas y estándares de la industria, especialmente si desde el *prompt* se solicita este factor. Además, en muchas ocasiones, fue más fácil resolver errores y problemas del código desarrollado por programadores humanos, siguiendo las recomendaciones que brindaban los modelos GPT. Esta tarea fue más rápida que consultando documentación y foros de discusión o bases de conocimiento como *Stackoverflow*[6].

La generación de elementos, como expresiones regulares para validar datos de entrada del sistema, fue más rápido y eficiente al obtenerlos del modelo GPT y después analizarlos y probarlos, que construyéndolos de manera tradicional. Asimismo, el modelo GPT puede generar el código de validaciones hechas a la medida para integrarlas en Laravel y agregarlas a la base de validaciones que ya tiene preconstruidas el *framework*.

individuales para utilizarlas en la medición y estimación de su trabajo.

5 Estimaciones basadas en *Personal Software Process* (PSP).

6 Stackoverflow – es una plataforma en Internet en dónde programadores publican preguntas y otros programadores publican las respuestas, constituyendo una base de conocimientos viva, en constante actualización.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 64 - 80

Mediciones de los componentes

La herramienta *Php Insights de Sail*, para análisis de código en proyectos PHP genera mediciones como la calidad del código (organización, legibilidad y coherencia del código), complejidad, arquitectura y estilo de codificación.

Aplicando la herramienta a algunos de los componentes que incluyen métodos con código adaptado obtenido de los modelos GPT, se obtuvieron métricas que se encuentran en semáforo verde y amarillo, como las que se muestra a continuación, y que permiten confirmar la validez y utilidad del código obtenido de los modelos:

Figura 4

Métricas del componente RegistrarCensoEthernetComponent.php

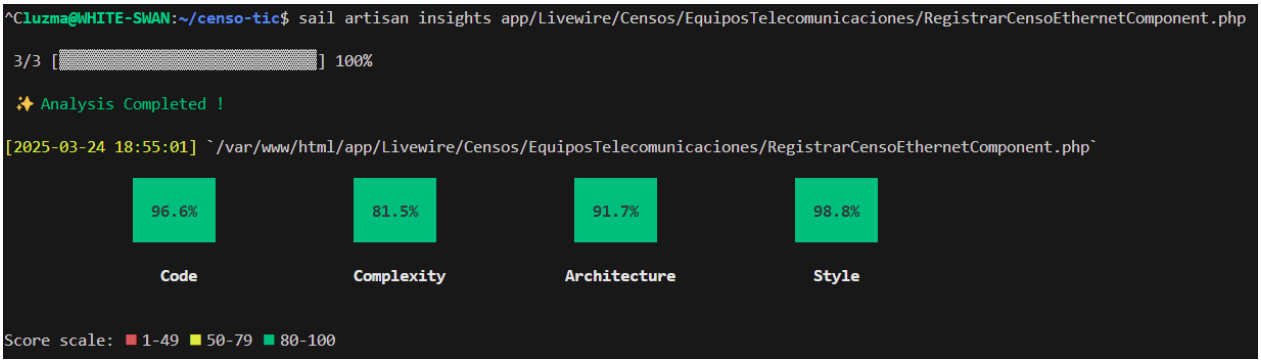


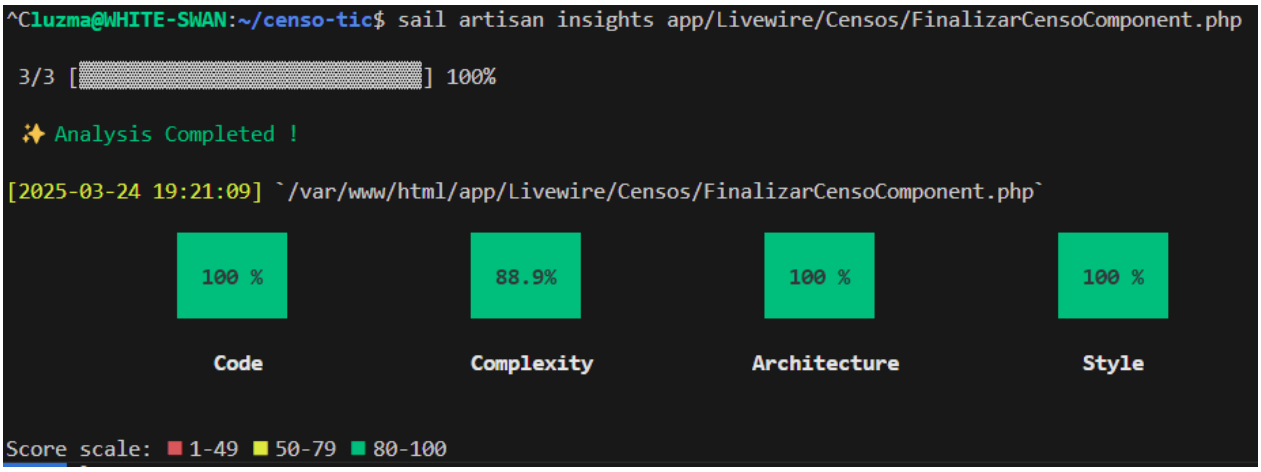
Figura 5

Métricas del componente RegistrarFibraOpticaComponent.php

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 65 - 80

Figura 6

Métricas del componente FinalizarCensoComponent.php



4. CONCLUSIONES

Dado que los beneficios que se obtuvieron en el desarrollo del Sistema de Censo utilizando modelos GPT también pueden alcanzarse en la construcción de cualquier software a la medida, se vuelve importante utilizar esta técnica en el desarrollo de software.

Al mismo tiempo, resulta imprescindible analizar y cuestionar el código que genera el modelo GPT ya que, en muchas ocasiones, el código generado no funcionará dentro del contexto del sistema en el que se está desarrollando la funcionalidad requerida.

El programador, que crea el *prompt* para generar código a través de los modelos GPT-4, debe contar con el marco de conocimientos técnicos, a fin de desarrollar un *prompt* específico, que brinde contexto y sea claro en el código que se espera, lo cual puede incluso involucrar el especificar tipos de datos de la entrada y la salida esperada. Mientras más información técnica brinde el *prompt*, el código generado será más cercano al requerimiento, de otra manera, se puede caer en muchas iteraciones, tratando de refinar el *prompt*, sin conseguir un resultado que funcione. Por esta razón, el programador debe seguir actualizándose y capacitándose, a fin de utilizar de manera eficiente los modelos de IA, en beneficio de la organización y de sus usuarios. Además, el programador analizará la eficiencia del código generado por el modelo de IA, a fin de obtenerla en el desempeño del sistema en el cuál se integrará para lo anterior, se requiere que el humano cuente con los conocimientos técnicos necesarios.

Los modelos GPT-4 ayudan a reducir los tiempos de desarrollo y corrección de incidencias desde el momento de escribir el código, ya que ponen automáticamente el nombre de las variables en las funciones, mismas que fueron declaradas como propiedades en las clases del componente, sugieren código (si uno declara un arreglo, el modelo sugiere el código para recorrerlo, o para desplegarlo en el blade) y ayudan en la creación de rutinas específicas, interrogando al modelo por medio de los *prompts*.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 66 - 80

Es importante resaltar que el ingenio de crear el *prompt* para solicitarle código al modelo GPT-4 pertenece al programador humano. En el desarrollo del *prompt*, se vierte la creatividad del programador para solicitar diferentes formas de construir el código que contribuya a resolver un problema de automatización y también se aplica el discernimiento humano para seleccionar la mejor solución generada y su adaptación para su integración al resto del sistema, como se ilustra en el Anexo 4.

El Anexo 5 es otra muestra de refinación del *prompt* con conocimientos técnicos, aunque, en este ejemplo, la mejor opción de código en cuanto a rendimiento de uso de memoria la brindó el modelo GPT. No obstante, es útil explorar estos cambios en el *prompt*, a fin de analizar diferentes caminos de resolver un mismo problema.

El proceso de aprendizaje y el entrenamiento de la mente humana para incrementar las capacidades lógica, analítica, creativa y orientadas a la solución de problemas debe ser un proceso permanente del ingeniero de software. También, en este proceso, los modelos GPT inciden, ya que es posible estudiar la explicación que brinda el modelo de cada solución que propone, pues en cada resultado generado por los modelos GPT, el programador humano puede aprender diferentes tácticas para resolver un mismo problema con el anhelado crecimiento profesional de cada programador humano.

REFERENCIAS

2024 Stack Overflow Developer Survey. (s.f). Recuperado el 15 de enero de 2025, de https://survey.stackoverflow.co/2024/

Brown, T., Mann, B., et al. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Degli-Esposti, S. (2023). *La ética de la inteligencia artificial*. Los Libros De La Catarata.

De, las H. del D., Rafael, Alonso, Á. G., & Carmen, L. G. (2018). *Métodos Ágiles. Scrum, Kanban, Lean*. Comercial Grupo ANAYA, S.A.

Dev, L. M. (2020, agosto 7). Return Early Pattern. *The Startup*. https://medium.com/swlh/return-early-pattern-3d18a41bba8

Developers warned: GitHub Copilot code may be licensed | TechTarget. (s/f). Search Software Quality. Recuperado el 24 de marzo de 2025, de https://www.techtarget.com/searchsoftwarequality/news/252526359/Developers-warned-GitHub-Copilot-code-may-be-licensed

iac. (2024, julio 24). *Microsoft Copilot con GitHub: ¿Cómo funciona?* Ingeniería Asistida Por Computador. https://www.iac.com.co/microsoft-copilot-con-github/

Jarrell, G. (s.f). *Chatgpt: Hacia Una Nueva Conciencia La Ingeniería De Prompts Para Chatgpt Que Va a Hacer De Ti El Genio Que Quieres Ser (La Superinteligencia Y Cómo Chatgpt Está Cambiando El Software La Comunicación Y La)*. Gordon Jarrell.

Javier Garzás en LinkedIn: ¿MIEDO subir tus datos a una IA? Pues instálate una en local, sin acceso… (s.f). Recuperado el 15 de enero de 2025, de https://es.linkedin.com/posts/jgarzas_miedo-subir-tus-datos-a-una-ia-pues-instalate-activity-7272954391580594176-uee_

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 67 - 80

Martí, M. A., Antonín, M. A. M., Llisterri, J., & Boix, J. L. (2002). *Tratamiento del lenguaje natural: Tecnología de la lengua oral y escrita*. Edicions Universitat Barcelona.

Martin, R. C. (2018). *Arquitectura limpia*. Editorial Anaya.

Martínez, R. N. (2015). *El Proceso de Desarrollo de Software*. IT Campus Academy.

Ramana, T. V., Ghantasala, G. S., Sathiyaraj, R., & Khan, M. (2024). *Artificial Intelligence and Machine Learning for Smart Community: Concepts and Applications*. CRC Press.

Reilly, E. (2024). *The No-Code Startup: The complete guide to building apps without code*. Practical Inspiration Publishing.

Robisco, S. (2024). *Historia de la Inteligencia Artificial*. Guadalmazán.

Solohubov, I., Moroz, A., Tiahunova, M. Y., Kyrychek, H. H., & Skrupsky, S. (s/f). *Accelerating software development with AI: exploring the impact of ChatGPT and GitHub Copilot*. CEUR Workshop Proceedings, 3679, 76-86. https://ceur-ws.org/Vol-3679/paper17.pdf

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 68 - 80

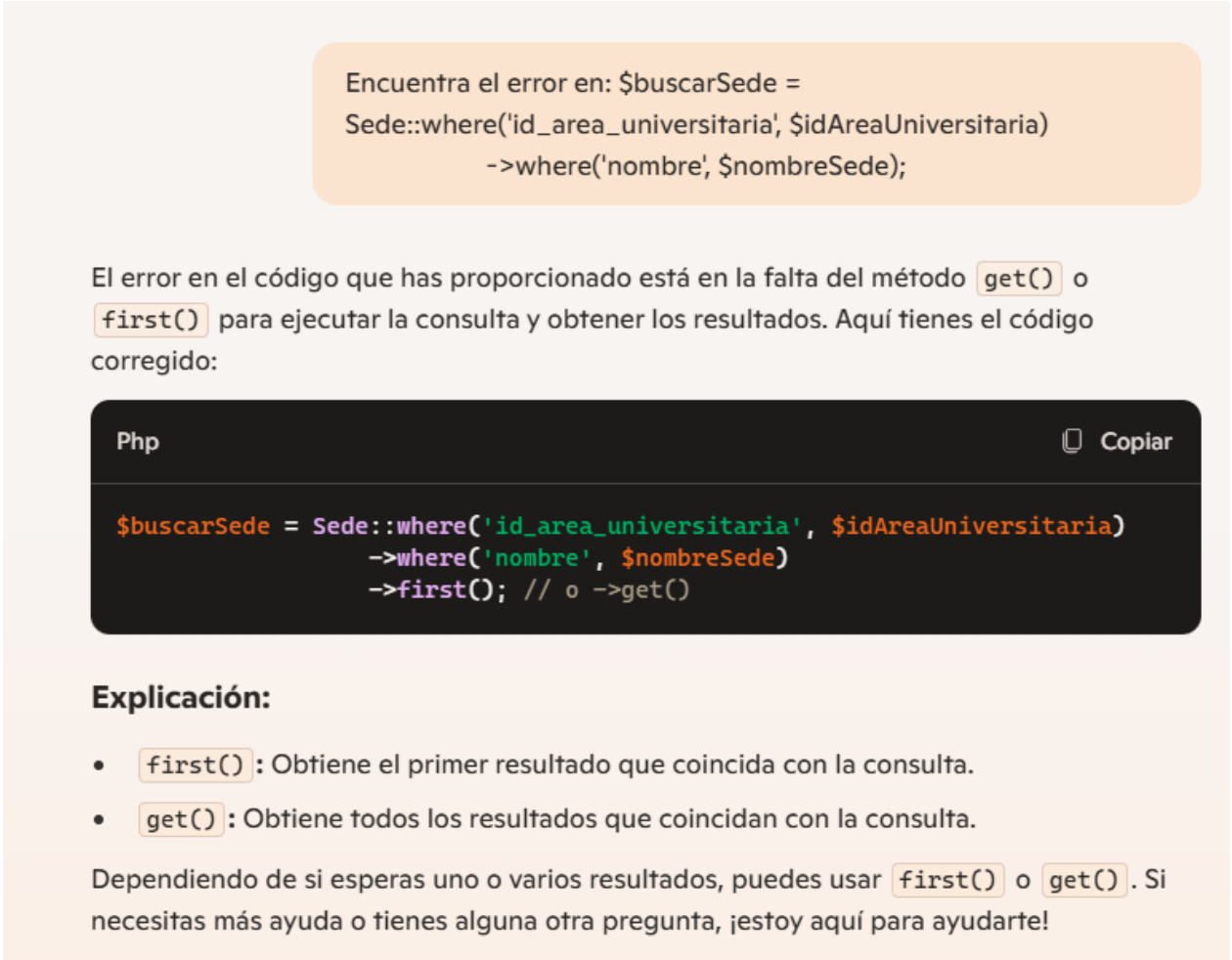
ANEXO 1

Ejemplo de correcciones de errores de sintaxis

En este ejemplo puede verse cómo el *prompt* es breve, con una tarea concreta y se debe contextualizar proporcionando el fragmento de código que causa el error, o bien, el fragmento de código que el programador detecta que tiene el problema, con ayuda de diferentes técnicas de depuración.

Figura 7

Prompt y respuesta del modelo GPT de ejemplo de corrección de errores de sintaxis



Encuentra el error en: $buscarSede = Sede::where('id_area_universitaria', $idAreaUniversitaria) ->where('nombre', $nombreSede);

El error en el código que has proporcionado está en la falta del método `get()` o `first()` para ejecutar la consulta y obtener los resultados. Aquí tienes el código corregido:

Php — Copiar

```php
$buscarSede = Sede::where('id_area_universitaria', $idAreaUniversitaria)
                  ->where('nombre', $nombreSede)
                  ->first(); // o ->get()
```

Explicación:

- `first()`: Obtiene el primer resultado que coincida con la consulta.
- `get()`: Obtiene todos los resultados que coincidan con la consulta.

Dependiendo de si esperas uno o varios resultados, puedes usar `first()` o `get()`. Si necesitas más ayuda o tienes alguna otra pregunta, ¡estoy aquí para ayudarte!

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 69 - 80

ANEXO 2

Ejemplo de generación de código para atender rutinas comunes

Figura 8

Prompt de ejemplo de generación de código para atender rutinas comunes

Genera en laravel un método para ordenar ascendentemente por id_herramienta, los elementos de un array asociativo que contiene id_herramienta, nombre, tipo_licenciamiento, proveedor.

En Laravel, puedes usar la función `usort()` de PHP para ordenar un array asociativo por `id_herramienta` de forma ascendente. Aquí tienes un método que lo hace dentro de una clase de servicio o controlador:

Php Copiar

```php
class HerramientaService
{
    public function ordenarPorIdHerramienta(array $herramientas): array
    {
        usort($herramientas, function ($a, $b) {
            return $a['id_herramienta'] <=> $b['id_herramienta'];
        });

        return $herramientas;
    }
}

// Ejemplo de uso
$herramientas = [
    ['id_herramienta' => 3, 'nombre' => 'Herramienta C', 'tipo_licenciamient
    ['id_herramienta' => 1, 'nombre' => 'Herramienta A', 'tipo_licenciamient
    ['id_herramienta' => 2, 'nombre' => 'Herramienta B', 'tipo_licenciamient
];

$service = new HerramientaService();
$herramientasOrdenadas = $service->ordenarPorIdHerramienta($herramientas);

print_r($herramientasOrdenadas);
```

Este método utiliza `usort()` y el operador de comparación `<=>`, lo que proporciona una ordenación eficiente y clara. ¿Quieres integrarlo en algún contexto específico dentro de Laravel, como una consulta de base de datos o un recurso de API? ¡Estoy aquí para ayudar! 🚀

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 70 - 80

ANEXO 3

Ejemplo de generación de consultas SQL

Figura 9

Prompt de ejemplo de generación de consultas SQL

Nota de la autora: El desarrollador debe adaptar el código puesto que seguramente las instrucciones con 'echo' no son necesarias en el contexto del sistema. También el parámetro $idCensoAntivirus debe pasarse de manera adecuada al método que se está implementando.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 71 - 80

ANEXO 4

Ejemplo 1 de búsqueda de alternativas de código

Figura 10

Prompt de ejemplo de búsqueda de alternativas de código

Nota de la autora:

Los desarrolladores en Laravel saben que hay métodos más eficientes para generar la consulta anterior que el primer resultado que ofreció el modelo GPT, así que se buscará refinar el *prompt* como sigue a continuación.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 72 - 80

Refinación del *prompt*, usando conocimientos técnicos

Figura 11

Ejemplo de refinación de prompt

Nota de la autora:

Se observa que, si el desarrollador brinda más detalles técnicos al modelo GPT, la respuesta del modelo mejora y se acerca a lo que espera el desarrollador humano.

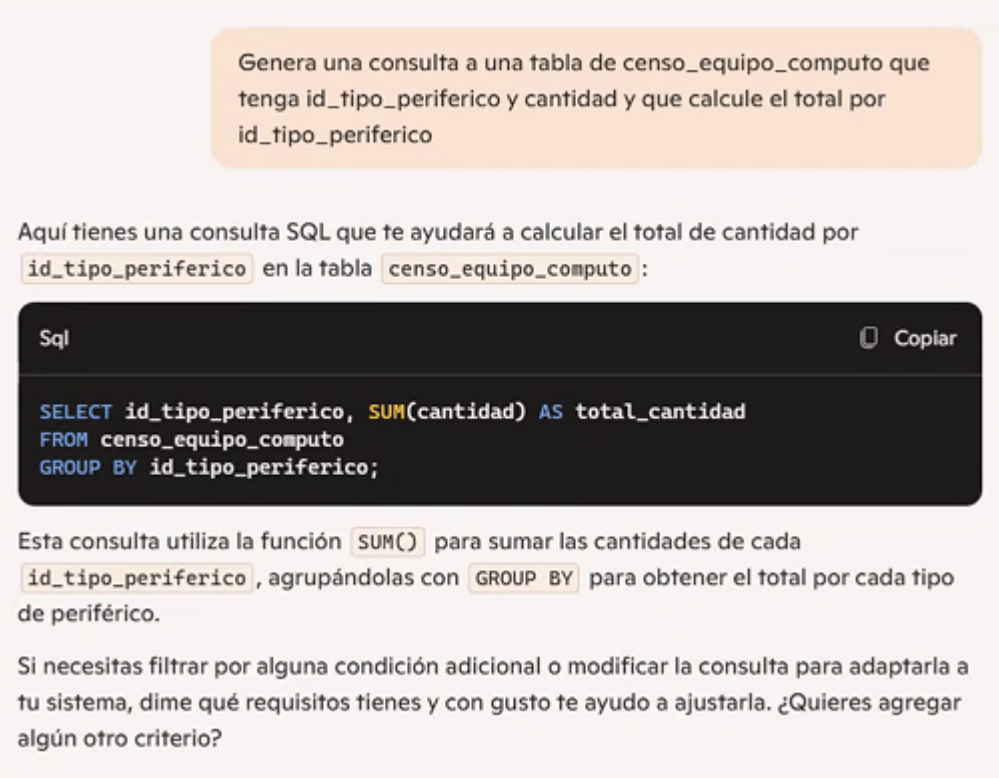
https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 73 - 80

ANEXO 5

Ejemplo 2 de búsqueda de alternativas de código

Figura 12

Prompt para solicitar una función de obtención de totales por tipo de periférico



https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 74 - 80

Figura 13

Prompt para la búsqueda de una alternativa con el método map

Nota de la autora:

En este ejemplo, el modelo GPT brindó de primera instancia la mejor solución. Se observa que el programador humano intentó refinar la respuesta a través del uso de otro método, sin embargo, según el modelo GPT, ésta no era la mejor opción en uso de recursos de memoria.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 75 - 80

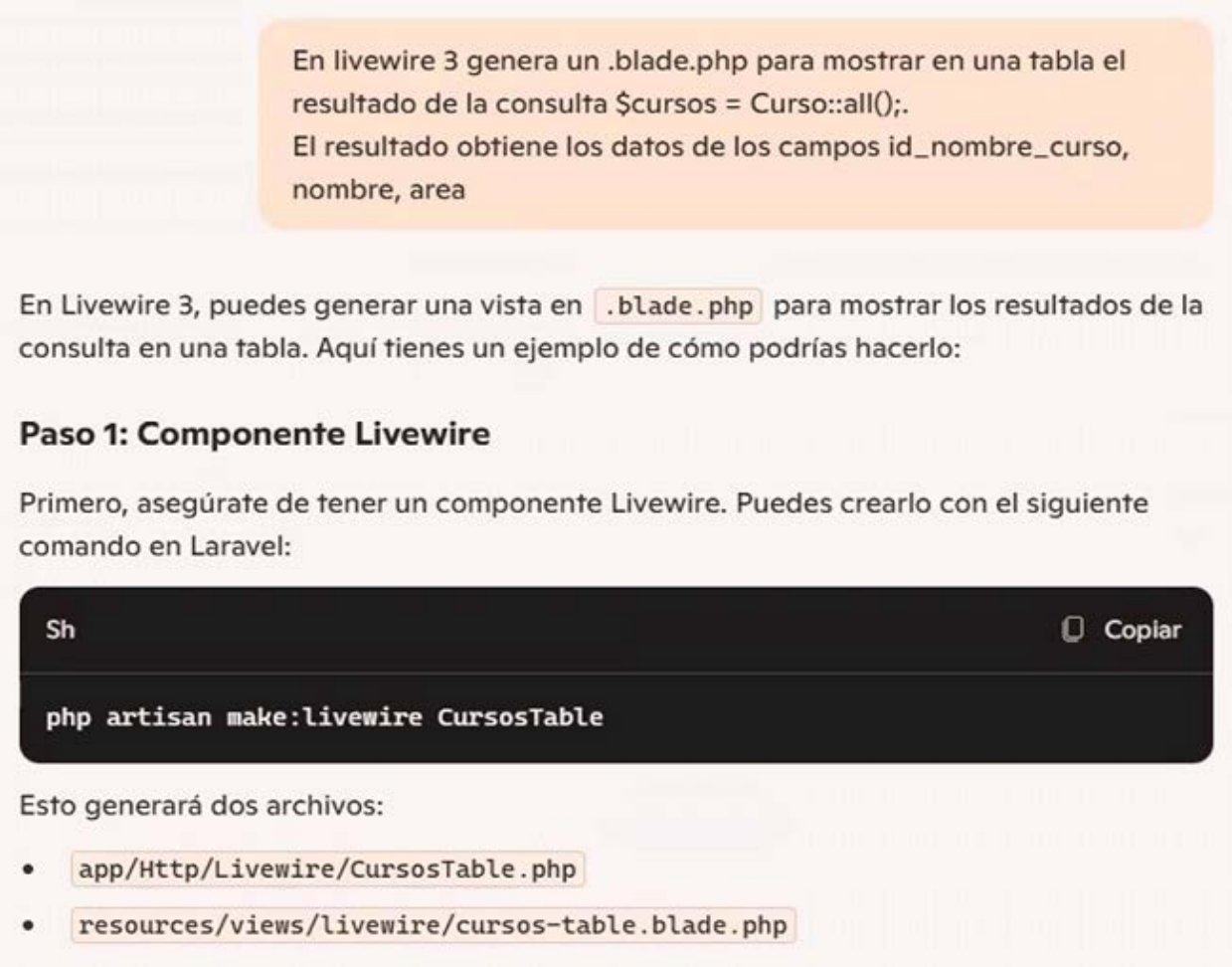
ANEXO 6

Generación de código para *front-end*

Las Figuras 14-A, 14-B y 14-C están relacionadas con el prompt solicitando la generación de un archivo de front-end (blade.php) para Livewire versión 3, incorporando una tabla html con el resultado de una consulta a una base de datos.

Figura 14-A

Primera parte del resultado del modelo ChatGPT para generar una tabla en un archivo .blade.php a partir de una consulta a un modelo de base de datos en Laravel-Livewire



https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 76 - 80

Figura 14-B

Segunda parte del resultado del modelo ChatGPT para generar una tabla en un archivo .blade.php a partir de una consulta a un modelo de base de datos en Laravel-Livewire

Paso 2: Controlador Livewire (`CursosTable.php`)

Edita el archivo `CursosTable.php` para obtener los cursos desde la base de datos:

Php Copiar

```php
namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Curso;

class CursosTable extends Component
{
    public $cursos;

    public function mount()
    {
        $this->cursos = Curso::all();
    }

    public function render()
    {
        return view('livewire.cursos-table');
    }
}
```

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 77 - 80

Figura 14-C

Tercera y última parte del resultado del modelo ChatGPT para generar una tabla en un archivo .blade.php a partir de una consulta a un modelo de base de datos en Laravel-Livewire

Paso 3: Vista Blade (`cursos-table.blade.php`)

Edita `cursos-table.blade.php` para mostrar los cursos en una tabla:

Blade — Copiar

```
<div>
    <table class="table-auto border-collapse border border-gray-400 w-full">
        <thead>
            <tr class="bg-gray-200">
                <th class="border border-gray-400 px-4 py-2">ID</th>
                <th class="border border-gray-400 px-4 py-2">Nombre</th>
                <th class="border border-gray-400 px-4 py-2">Área</th>
            </tr>
        </thead>
        <tbody>
            @foreach($cursos as $curso)
                <tr>
                    <td class="border border-gray-400 px-4 py-2">{{ $curso->
                    <td class="border border-gray-400 px-4 py-2">{{ $curso->
                    <td class="border border-gray-400 px-4 py-2">{{ $curso->
                </tr>
            @endforeach
        </tbody>
    </table>
</div>
```

Paso 4: Incluir el Componente en la Vista Principal

Para mostrar la tabla en alguna vista de Laravel, incluye el siguiente código en `welcome.blade.php` o en la vista donde lo necesites:

Blade — Copiar

```
@livewire('cursos-table')
```

Nota de la autora:

En este ejemplo, se observa que el modelo GPT no sólo generó el archivo .blade.php solicitado. Además, sugirió el contenido del componente de Livewire, con la consulta a la base de datos y mostró cómo incluir el componente de livewire en una vista del framework Laravel, con el que trabaja Livewire.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 78 - 80

ANEXO 7

Búsqueda de librerías de código

Figura 15

Prompt solicitando librerías para generar gráficas

Nota de la autora:

Sólo se incluyó la respuesta completa del modelo GPT en la primera librería recomendada para no hacer muy extenso el texto. En cada recomendación de librerías, el modelo GPT incluyó información sobre cómo hacer su instalación y configuración, así como un ejemplo de uso para hacer más fácil el proceso de selección de la librería. Es de resaltar que toda la información está recopilada en la respuesta del modelo GPT, lo que ahorra tiempo en la búsqueda de la información y de los ejemplos de uso en la documentación técnica de cada librería.

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 79 - 80

ANEXO 8

Acortar la curva de aprendizaje

Las Figuras 16-A y 16-B están relacionadas con el *prompt* para el aprendizaje del uso de #[On para el manejo de eventos en la nueva versión de livewire (3) para el desarrollo de aplicaciones a la medida

Figura 16-A

Primera parte de la respuesta del modelo GPT para el aprendizaje del uso de #[On

https://doi.org/10.22201/dgtic.30618096e.2025.3.2.111 **Vol. 3, Núm. 2.** abril-junio 2025, págs. 80 - 80

Figura 16-B

Segunda parte de la respuesta del modelo GPT (paso de parámetros) para concluir la explicación del uso de #[On

🔹 **Pasar parámetros**

Si el evento envía datos adicionales, estos se pueden recibir en el método:

Php — Copiar

```php
use Livewire\Component;
use Livewire\Attributes\On;

class Dashboard extends Component
{
    #[On('post-created')]
    public function updatePostList($title)
    {
        // Actualizar la lista con el nuevo título...
    }
}
```

🔹 **Escuchar eventos dinámicos**

También puedes **dinamizar** los nombres de los eventos, por ejemplo, si quieres que un evento solo afecte a un modelo específico:

Php — Copiar

```php
use Livewire\Component;

class UpdatePost extends Component
{
    public function update()
    {
        $this->dispatch("post-updated.{$post->id}");
    }
}
```

Y luego escucharlo en otro componente:

Php — Copiar

```php
use Livewire\Component;
use App\Models\Post;
use Livewire\Attributes\On;

class ShowPost extends Component
{
    public Post $post;

    #[On('post-updated.{post.id}')]
    public function refreshPost()
    {
        // Código para actualizar el post...
    }
}
```

Nota de la autora:

La información y ejemplos del uso de la directiva #[On está concentrada en la respuesta del modelo para facilidad de quién estudia esta actualización al *framework* Livewire en su versión 3 .