ChatGPT

Métodos alternativos para la generación de documentos a partir de plantillas (PDF/Excel)

Introducción

La generación automatizada de documentos a partir de plantillas (similar a la función de AppSheet) consiste en tomar un **documento plantilla** con *placeholders* o marcadores de posición y reemplazarlos por valores dinámicos proporcionados como parámetros. Esto permite crear PDFs, hojas de cálculo u otros formatos con datos personalizados sin tener que editar manualmente cada documento. A continuación, se exploran diversas alternativas – desde bibliotecas de programación (por ejemplo, en Python u otros lenguajes) hasta herramientas y servicios especializados – para lograr esta generación de documentos de forma más flexible, superando las limitaciones que presenta AppSheet. Dado el contexto empresarial y la confidencialidad de los datos, se hará énfasis en soluciones seguras y controlables dentro de la propia infraestructura.

Limitaciones de la generación de documentos con AppSheet

AppSheet (plataforma de Google) permite generar documentos automáticamente usando plantillas en Google Docs, Microsoft Word, Excel, HTML, etc., llenándolas con datos de la app. Sin embargo, esta funcionalidad presenta **varias limitaciones** importantes:

- **Formateo limitado en plantillas de Excel**: AppSheet solo soporta plantillas de Excel sencillas. Por ejemplo, **no admite tablas ni tablas dinámicas (pivot tables) en un template de Excel**, y no permite tablas anidadas en la misma hoja [1]. Esto impide crear documentos de Excel con estructuras complejas mediante AppSheet.

- **Limitaciones en PDFs generados desde Google Docs**: Al usar una plantilla de Google Docs para generar PDF, muchos elementos de formato avanzado no se respetan. Usuarios han reportado la falta de **encabezados, pies de página, marcas de agua, control de orientación de página y saltos de página apropiados** en los PDFs generados por AppSheet [2]. En otras palabras, el motor de renderizado de AppSheet para PDFs es básico y no soporta ciertas características que sí son posibles manualmente o con Apps Script.

Estas restricciones hacen que los documentos producidos con AppSheet a veces no cumplan con los requisitos de presentación o complejidad necesarios. Por ello, empresas buscan **alternativas** que les den mayor control sobre el formato y puedan manejar casos más avanzados, manteniendo a la vez la automatización.

Soluciones basadas en programación (código)

Una forma de superar las limitaciones de AppSheet es implementar la generación de documentos mediante código propio, usando lenguajes tradicionales como **Python, JavaScript (Node.js), C#/.NET, Java**, etc. Estas soluciones permiten mayor flexibilidad y control. A continuación se describen varios métodos programáticos:

Generación de PDF mediante código

- **Bibliotecas para construir PDF "desde cero"**: Un enfoque es escribir código para dibujar o escribir contenido en un PDF directamente. Por ejemplo, la librería **FPDF** (disponible en Python y PHP) o **ReportLab** (Python) permiten crear PDFs programáticamente definiendo textos, imágenes, tablas, etc. Este método ofrece **control total del diseño**, pero requiere más trabajo manual (codificar posiciones, fuentes, etc.). En un artículo técnico se menciona esta aproximación como "usar la biblioteca FPDF en Python para dibujar directamente un PDF" [3]. La ventaja es que no dependemos de plantillas preexistentes, pero puede ser tedioso para documentos complejos o con mucho formato fijo.

- **Plantillas HTML + conversión a PDF**: Otra estrategia popular es generar primero un **HTML con los datos** y luego convertir ese HTML a PDF. Se utiliza un motor de plantillas (por ejemplo **Jinja2** en Python) para crear una página HTML con placeholders reemplazados, y después se transforma a PDF con herramientas como **wkhtmltopdf** (a través de un wrapper como `pdfkit`) o bibliotecas como **WeasyPrint** o **xhtml2pdf** [4] [5]. Este método permite diseñar el documento usando HTML/CSS (facilitando el estilo) y lograr un PDF final que respete ese diseño. Por ejemplo, se puede definir una plantilla HTML de factura o reporte con etiquetas Jinja2 (`{{ }}`) y el código Python rellena esa plantilla con datos; luego con WeasyPrint o wkhtmltopdf se obtiene el PDF resultante [6]. Esto combina facilidad de diseño (HTML es más accesible que las APIs de PDF puras) con automatización. Es importante notar que la fidelidad del PDF al HTML depende del motor de conversión; wkhtmltopdf (basado en WebKit) suele ofrecer resultados precisos [7].

- **Plantillas PDF con campos rellenables**: Menos común pero posible es tener un PDF existente con **campos de formulario** o marcadores de texto específicos, y usar código para insertar datos allí. Por ejemplo, con bibliotecas Python como PyPDF2, pdfrw o PyMuPDF se pueden modificar PDFs existentes para rellenar campos. Este enfoque aprovecha un PDF pre-diseñado (formato fijo) donde solo se completan espacios vacíos. Sin embargo, requiere que el PDF original tenga formularios definidos o texto identificable a buscar y reemplazar.

Plantillas de documentos Office (Word) y conversión a PDF

- **Plantillas Word (DOCX) con motor de "mail merge" o similar**: Una solución robusta es usar documentos Word (.docx) como plantillas con marcadores, y un código que realice la "combinar correspondencia" de forma automática. En Python existe la librería **python-docx-template (docxtpl)** que permite cargar un .docx con sintaxis Jinja2 incrustada (por ejemplo, escribir en Word `{{nombre_cliente}}` donde deba ir un nombre) y luego desde Python pasar un diccionario de datos para renderizar un nuevo .docx con esos valores [8] [9]. Docxtpl utiliza internamente `python-docx` y Jinja2 [10], facilitando incluso lógica en la plantilla (loops para tablas repetitivas, condiciones, inserción de imágenes, etc.). Una vez generado el `.docx` final, se puede **convertirlo a**

PDF si se desea – por ejemplo, usando **LibreOffice/OpenOffice en modo headless**, la API de Word (en entornos .NET/COM) o bibliotecas dedicadas. Un artículo técnico destaca esta técnica como muy flexible: diseñadores no programadores pueden crear la plantilla en Word, y el desarrollador solo provee los datos y ejecuta la fusión [11] [12]. De hecho, librerías comerciales como **Aspose.Words** ofrecen funciones de Mail Merge similares: se prepara un documento Word con campos de combinación, luego el código carga ese template y une los datos produciendo un documento final (Word o PDF) [13].

- **Bibliotecas en .NET/Java para Word**: En entornos empresariales, es común el uso de componentes comerciales robustos. Por ejemplo, **Aspose.Words** (disponible para .NET, Java y también usable en Python vía .NET) permite cargar plantillas de Word o incluso PDF y rellenarlas con datos usando sintaxis especial (por ejemplo *linq reporting* o campos MERGEFIELD) [13]. Otra opción es **Open XML SDK** de Microsoft (para manipular .docx, .xlsx directamente en .NET), o herramientas como **Windward Studios** que integran un motor de reporte donde las plantillas se diseñan en Word usando etiquetas y luego se generan documentos mediante un API. Estas bibliotecas suelen manejar imágenes, tablas y formatos complejos con facilidad, a cambio de ser soluciones de pago en muchos casos.

Generación de hojas de cálculo Excel a partir de plantillas

- **Uso de bibliotecas Python para Excel**: Para generar reportes en Excel (`.xlsx`) con datos dinámicos, se puede adoptar un enfoque similar al de Word. Por ejemplo, con **OpenPyXL** o **XlsxWriter** se pueden crear o modificar hojas Excel mediante código. Un método práctico es diseñar una **plantilla Excel con marcadores de texto** (por ejemplo escribir `{{cliente}}` en una celda donde deba ir el nombre del cliente) y luego usar Python para abrir esa plantilla y reemplazar esos placeholders por los valores reales. Esto se logra recorriendo las celdas y haciendo búsqueda y reemplazo de cadenas. Un tutorial ilustra este proceso: se crea un Excel `template.xlsx` con textos como `{{ placeholder }}`, luego el código con OpenPyXL carga el libro, recorre cada celda buscando esos patrones `{{...}}` y los sustituye por los datos proporcionados, guardando un nuevo Excel resultante [14] [15]. En el ejemplo citado, se rellena la plantilla reemplazando `{{ placeholder }}` por "Hello, Python Excel Automation!" automáticamente [14] [15]. Tras la sustitución, el script guarda el archivo final (`output.xlsx`). Este enfoque conserva todo el formato original de la plantilla (estilos, fórmulas, gráficos predefinidos, etc.) y solo cambia los textos marcados. De forma similar, se pueden generar **múltiples filas o tablas dinámicamente**: por ejemplo, poniendo en la plantilla una fila de ejemplo con marcadores y duplicándola en código por cada registro necesario.

- **Librerías y módulos especializados**: Existen también librerías diseñadas específicamente para plantillas Excel. En Node.js, por ejemplo, está **xlsx-template** (módulo npm) que facilita la sustitución de marcadores en archivos .xlsx [16]. Otra herramienta es **Docxtemplater** (JS), que no se limita a Word sino que soporta plantillas Excel y PowerPoint utilizando sintaxis de etiquetas similar (`{nombre}` para texto, y construcciones `{#loops}{/loops}` para listas) [17]. Estas bibliotecas permiten usar Excel mismo como diseñador de la plantilla, lo que significa que cualquier persona familiarizada con Excel puede preparar el formato deseado con sus fórmulas, estilos y simplemente colocar texto clave donde van los datos variables [18] [19]. Luego la librería se encarga de leer ese archivo y reemplazar los marcadores por los datos. *Pros:* se aprovechan todas las capacidades nativas de formato de Excel y se reduce la complejidad del código (no hay que programar la

estructura del Excel, solo la sustitución) [20] [21]. *Contras:* la flexibilidad depende de la plantilla; si se requieren cambios estructurales grandes habría que modificar la plantilla manualmente.

- **Generación desde cero**: Como alternativa a usar una plantilla estática, siempre existe la opción de construir la hoja Excel completamente con código (ejemplo: usar **pandas** para volcar datos y **OpenPyXL/XlsxWriter** para formatear). Esto brinda más control algorítmico (p.ej. crear número variable de columnas o hojas según los datos). Sin embargo, es más complejo dar formato elaborado por código que aprovechar una plantilla preformateada. Un buen equilibrio es mezclar enfoques: usar código para las partes dinámicas (por ejemplo agregar filas) pero apoyarse en plantillas para estilos base o partes fijas.

Tecnologías y lenguajes adicionales

- **JavaScript/Node.js**: Ya mencionado en parte con Docxtemplater para Office, el ecosistema JS también tiene opciones para PDF. Por ejemplo, **pdfmake** o **jsPDF** permiten construir PDFs (en navegador o Node). En servidores Node, además de docxtemplater para Office, hay paquetes para PDF similares a los de Python (por ejemplo, usando Puppeteer o Headless Chrome para convertir HTML a PDF). Si ya se trabaja en un stack JavaScript, estas herramientas evitan tener que invocar Python o Java.

- **Java/Scala**: En Java el uso de **iText (iText7)** es muy difundido para generar PDF a bajo nivel. Para documentos de Word/Excel, Java cuenta con **Apache POI** (manipula formatos Office) con el cual se puede, por ejemplo, abrir un .xlsx como plantilla y editar celdas. También hay soluciones como **Docx4j** (para Word en Java) y librerías de mail merge en Word (por ej. **JasperReports** puede combinar datos con templates ODT/DOCX). Muchas de estas requieren escribir bastante código de manejo de documentos XML subyacente, pero funcionan on-premise sin servicios externos.

- **.NET (C#)**: En .NET existe **Open XML SDK** de Microsoft para Word/Excel, así como bibliotecas open source (p.ej. **EPPlus** para Excel) que permiten crear o modificar estos archivos en servidores Windows o Linux (vía .NET Core). Para PDF, iText tiene versión .NET (iTextSharp), y también **PdfSharp** o **Syncfusion PDF**. Además, .NET puede automatizar Office de forma "headless" aunque no es recomendado para servidores. Una ventaja de .NET es la posibilidad de usar **Office Interop** o **Mail Merge** nativo de Word si la aplicación corre en un entorno con Word instalado, pero nuevamente eso es más propio de escritorio que de servidores.

En resumen, las alternativas programáticas ofrecen: **flexibilidad** (se pueden implementar prácticamente cualquier reemplazo o formato deseado), **independencia** de proveedores (el código es propio) y la posibilidad de integrar la generación de documentos en sistemas internos. El costo es la inversión de desarrollo y mantenimiento de ese código. Para organizaciones con equipo técnico, valerse de bibliotecas como las mencionadas (Jinja2, OpenPyXL, docxtpl, etc.) puede solventar las limitaciones de AppSheet proporcionando documentos más ricos en formato y sin las restricciones que aquella impone.

Herramientas y servicios especializados en generación de documentos

Además del código "propio", existen **plataformas y servicios dedicados** a la generación de documentos a partir de plantillas. Algunas son productos comerciales (SaaS o instalables) que simplifican la tarea ofreciendo editores de plantillas y APIs para fusionar datos, con enfoque empresarial. Estas soluciones suelen tener componentes visuales para diseñar documentos y se integran vía API o conectores con las fuentes de datos.

- **Motores de plantillas self-hosted o SaaS**: Un ejemplo destacado es **Docmosis**, que ofrece un motor de documentos basado en plantillas de Word/LibreOffice. El usuario diseña la plantilla en su procesador de texto (usando marcadores de texto plano como `<<Nombre>>` por ejemplo), y Docmosis permite mediante una **simple API REST** enviar los datos (en JSON, XML, etc.) y recibir el documento generado [22] [23]. Soporta lógica compleja en las plantillas (condicionales, bucles para listas repetitivas, cálculos, formato numérico) utilizando marcadores especiales, todo dentro del propio Word. Docmosis puede **desplegarse tanto en la nube como on-premises** según la necesidad [24] [25]. La ventaja es que usuarios no técnicos pueden mantener las plantillas en Word, mientras los desarrolladores solo se preocupan de llamar al servicio con los datos. Otros motores similares incluyen **Windward Studios**, **Telerik Reporting**, **Spire/HotDocs**, etc., que generalmente permiten diseñar plantillas en Word o diseñadores visuales e integrarse con aplicaciones.

- **Servicios web de generación de documentos (Document Generation SaaS)**: Hay soluciones 100% en la nube como **Formstack Documents** (antes WebMerge), **PandaDoc**, **DocuSign Gen**, **Plumsail Documents** (en entorno Microsoft) y **PDF Generator API**, entre otros. Estos servicios ofrecen interfaces para crear plantillas (a veces con editores drag-and-drop o aprovechando archivos Word/PDF de base) y exponen una API para subir datos y obtener documentos. Por ejemplo, **PDF Generator API** permite generar PDFs a partir de plantillas y datos JSON, e incluso provee un editor de plantillas online; además, ofrece opción de **despliegue on-premises** para clientes empresariales que necesiten mayor control [26] [27]. La ventaja de SaaS es la rapidez de implementación (no hay que reinventar el motor de documentos), pero se debe analizar la privacidad al enviar datos a la nube (ver sección de seguridad abajo).

- **Automatización con herramientas de flujo (no-code/low-code)**: Si se busca una vía sin código tradicional, pero fuera de AppSheet, se puede usar plataformas de automatización. En el ecosistema Microsoft, por ejemplo, **Power Automate** tiene acciones para **"Rellenar una plantilla de Word"**: uno prepara un .docx con controles de contenido o marcadores, y el flujo los llena con campos de una lista o formulario, pudiendo generar luego PDF o DOCX de salida [28] [29]. También existe **SharePoint Syntex** (Content Assembly) que permite diseñar plantillas modernas y poblarlas vía flujos de Power Automate [30] [31]. Este tipo de soluciones permiten a analistas de negocio automatizar documentos integrándose con datos de SharePoint, SQL, etc., sin programación. En entorno Google, se puede recurrir a **Google Apps Script**: un script puede tomar un Google Doc de plantilla y hacer búsqueda/reemplazo de placeholders, generando luego un PDF. De hecho, muchas implementaciones utilizan Apps Script para lograr lo que AppSheet limita – es posible copiar un documento plantilla y sustituir tags como `<<Cliente>>` con valores desde una hoja de cálculo, luego exportar a PDF automáticamente [32] [33] (Apps Script no tiene esas limitaciones de formato, por lo que puede respetar encabezados, saltos de página, etc., que AppSheet omite). Herramientas

de terceros como **Zapier** o **Integromat/Make** también ofrecen integraciones predefinidas para tomar datos de un formulario o base de datos y combinarlos en una plantilla Google Doc/Word.

En general, las herramientas especializadas buscan **facilitar la creación de documentos complejos sin tener que escribir todo el código desde cero**. Suelen incorporar características como: editores visuales de plantillas, integración con múltiples fuentes de datos, salidas a varios formatos (PDF, Word, Excel, HTML, etc.) [22] [23], y manejo de volúmenes altos con colas o lotes. Para una empresa que requiera alta confiabilidad y soporte, puede ser conveniente considerar estos productos, evaluando costo-beneficio frente a desarrollar internamente la solución.

Consideraciones de seguridad y confidencialidad

Dado que en el caso planteado los documentos contendrán **datos empresariales confidenciales**, es crucial evaluar el aspecto de seguridad en cualquier método elegido:

- **Ejecutar la generación in-house vs. nube**: Mantener el proceso dentro de la infraestructura de la empresa brinda mayor control de los datos. Por ejemplo, usar bibliotecas de Python/Java en un servidor propio o adoptar una solución on-premises significa que la información sensible nunca sale al exterior, lo cual maximiza la privacidad [34]. En cambio, si se utiliza un servicio en la nube (SaaS) para generar documentos, los datos se transmitirán a ese servicio; en ese caso, es vital confiar en el proveedor, asegurar que haya cifrado en tránsito y en reposo, y posiblemente firmar acuerdos de procesamiento de datos (DPA) con ellos. Algunas plataformas SaaS ofrecen la alternativa de despliegue **auto-hospedado** precisamente para clientes con altos requerimientos de seguridad [26].

- **Control de accesos y almacenamiento**: Sea cual sea la solución, se deben implementar controles para que solo personal o sistemas autorizados puedan acceder a las plantillas y a los documentos generados. Las plantillas mismas pueden contener información sensible (por ejemplo, el formato de un contrato confidencial), por lo que deben almacenarse en repositorios seguros. Igualmente, los documentos resultantes quizá deban **cifrarse** si se guardan temporalmente en algún lugar. Si la generación es con código propio, asegúrese de no dejar archivos temporales con datos sensibles sin protección y de eliminar/limpiar variables en memoria según corresponda.

- **Prevención de inyección de contenido no deseado**: Cuando se usan motores de plantillas (especialmente aquellos que ejecutan lógica, como Jinja2), hay que considerar la sanitización de los datos de entrada para evitar que contenido malicioso se inserte en el documento. En entornos controlados internos esto suele ser menos preocupante (los datos provienen de sistemas confiables), pero es una buena práctica. Muchas bibliotecas templating (Jinja2, etc.) tienen *sandbox* o modos seguros para evitar ejecución de código arbitrario en las plantillas [35] [36].

- **Cumplimiento y trazabilidad**: En contextos empresariales, puede ser necesario **auditar** el proceso de generación de documentos. Por ejemplo, llevar un registro de qué datos se insertaron en qué documento y cuándo. Si se construye la solución con código, se puede agregar logging detallado. Si se usa un servicio externo, verificar si ofrece logs de documentos generados o watermarks de seguridad.

En resumen, **la opción más segura** suele ser aquella donde **los datos no abandonan el entorno corporativo**. Si la empresa cuenta con equipo de desarrollo, implementar una solución interna con bibliotecas como las mencionadas (o usar un producto on-premises) garantiza confidencialidad total de los datos [34]. Si se opta por un servicio externo por su conveniencia, habrá que reforzar los acuerdos de seguridad y quizá anonimizar o cifrar parcialmente datos sensibles antes de enviarlos, dependiendo de la sensibilidad.

Conclusión

Existen múltiples caminos para lograr la generación automatizada de documentos PDF y Excel a partir de plantillas, evitando las limitaciones de AppSheet. Las alternativas van desde **programar la solución** con herramientas robustas (ejemplos: generar HTML y convertir a PDF con Python [4] [5], o usar plantillas .docx/.xlsx con código para rellenarlas [15] [37]), hasta emplear **plataformas especializadas** que simplifican el trabajo (motores de plantillas empresariales, servicios API de documentos, o flujos no-code en otras plataformas). La elección depende de varios factores: la complejidad de los documentos, los recursos técnicos disponibles, el presupuesto y, muy importante, los **requisitos de seguridad** de la información que contendrán. En un entorno empresarial con datos confidenciales, suele preferirse una solución donde el procesamiento ocurra **bajo su control directo (on-premise)** para garantizar la privacidad [34]. No obstante, incluso soluciones en la nube pueden configurarse de forma segura y ofrecer gran agilidad.

En última instancia, todas las opciones comparten la misma idea fundamental: tener una plantilla con marcadores y un mecanismo para combinarlos con los datos. Ya sea escribiendo código con Python u otro lenguaje, o utilizando una herramienta dedicada, es posible lograr documentos dinámicos **personalizados**, con formato profesional, superando los límites que impone AppSheet en su generación nativa. Lo importante es evaluar las necesidades específicas (formato, volumen, seguridad) y optar por la alternativa que mejor se adapte, sabiendo que **hay disponible un amplio abanico de tecnologías** para llevar a cabo esta tarea de manera eficiente y confiable.

Referencias: Las afirmaciones y métodos descritos se basan en documentación técnica y ejemplos de las herramientas mencionadas, incluyendo: limitaciones oficiales de AppSheet [1] [2], guías sobre generación de PDF con Python (Jinja2 + pdfkit) [4] [5], artículos sobre automatización con docxtpl y FPDF [3], métodos de relleno de plantillas Excel con OpenPyXL [15] [37], información de productos como Docmosis [22] [23], Aspose.Words [13], y consideraciones de despliegue on-premise para seguridad [34] [26], entre otros. Cada solución debe estudiarse a detalle para su correcta implementación, pero todas apuntan al objetivo común de generar documentos empresariales de forma automática, flexible y segura.

---

[1] Use Microsoft Excel templates - AppSheet Help
https://support.google.com/appsheet/answer/11575067?hl=en

[2] [32] [33] PDF Limitations on Google Docs Templates - AppSheet Q&A - Google Developer forums
https://discuss.google.dev/t/pdf-limitations-on-google-docs-templates/293210

[3] [8] [9] [10] How to automate creating reports using docx templates | by Holistic AI Engineering | Medium
https://medium.com/@engineering_holistic_ai/how-to-automate-creating-reports-using-docx-templates-bc3cbaae069e

4 5 6 7 35 36 Create PDF reports with Python and Jinja2 - lin^2
https://linlinzhao.com/tech/2021/01/20/jinja-report.html

11 12 17 Docxtemplater | Word, Powerpoint, Excel generation using templates in your application | docxtemplater
https://docxtemplater.com/

13 Crear PDF Informe En Python
https://products.aspose.com/words/es/python-net/report/pdf/

14 15 16 18 19 20 21 37 7 Key Elements of Effective Microsoft Excel Automation with Python
https://www.softkraft.co/python-excel-automation/

22 23 24 25 Document Generation - Docmosis
https://www.docmosis.com/

26 27 Secure and Private On-premises Deployment - PDF Generator API
https://pdfgeneratorapi.com/on-premises-deployment

28 29 30 31 Automate document generation with document processing and Power Automate | Microsoft Learn
https://learn.microsoft.com/en-us/microsoft-365/documentprocessing/automate-document-generation?view=o365-worldwide

34 Document generation software: A complete guide in 2024 - Oneflow
https://oneflow.com/blog/document-generation-software-a-complete-guide/