Automatización de Procesos: Desarrollo de un Aplicativo para Optimizar Tareas Administrativas

M. Benítez Soler

Resumen— Este proyecto presenta el desarrollo de un aplicativo web diseñado para automatizar tareas administrativas repetitivas relacionadas con el mundo de las auditorías de riesgo tecnológico, específicamente aquellas que requieren rellenar una serie de formularios a través de la información contenida en un Excel previamente respondido por cliente. El aplicativo permite al usuario subir dicho Excel con todas las respuestas contenidas y empezar el proceso de automatización después de un solo clic. Dicho proceso extrae los datos necesarios del archivo, organiza las respuestas, y muestra los formularios totalmente rellenados, permitiendo al usuario comprobar toda la información y corregir las respuestas que considere necesarias. El desarrollo se ha dividido en tres fases principales: creación de la interfaz, construcción de los formularios y respuestas, y automatización del proceso a través de la lógica del servidor. Los resultados obtenidos muestran una mejora significativa en cuanto a eficiencia, reduciendo el tiempo de realización de la tarea hasta 10 veces, además de eliminar posibles errores humanos.

Palabras clave— Automatización, Formularios web, Node.js, Excel, Auditoría, Riesgo IT, Vue.js, Express.js, JavaScript, ISA 315, Procesos Administrativos, HTML, CSS, Interfaz de Usuario.

Abstract— This project presents the development of a web application designed to automate repetitive administrative tasks related to the world of technological risk audits, specifically those that require filling out a series of forms through the information contained in an Excel previously answered by the client. The application allows the user to upload the Excel file with all the answers and start the automation process after a single click. This process extracts the necessary data from the file, organizes the answers, and displays the forms completely filled in, allowing the user to check all the information and correct the answers he/she considers necessary. The development has been divided into three main phases: creation of the interface, construction of the forms and answers, and automation of the process through the server logic. The results obtained show a significant improvement in terms of efficiency, reducing the time to complete the task up to 10 times, as well as eliminating possible human errors.

Index Terms— Automation, Web Forms, Node.js, Excel, Audit, Technology Risk, Vue.js, Express.js, JavaScript, ISA 315, Administrative Processes, HTML, CSS, User Interface.

1 Introducción

En un mundo cada vez más digitalizado, la gestión eficiente de los procesos administrativos se ha convertido en un factor clave para mejorar la eficiencia y reducir la carga de trabajo manual.

Este trabajo surge en el marco de una experiencia profesional en el equipo Technology Risk de EY, donde se realizan auditorías asociadas a los aplicativos IT de diferentes empresas.

En el equipo de auditoría Technology Risk nos encargamos de evaluar, analizar y mitigar los riesgos asociados a los sistemas y aplicaciones IT dentro de cada organización, con el objetivo de garantizar la seguridad, fiabilidad y cumplimiento de la normativa de los sistemas tecnológicos utilizados por cada empresa. Se realizan controles internos, se identifican vulnerabilidades y se evalúan los posibles riesgos que puedan impactar la operabilidad y cumplimiento regulatorio.

Como parte de este proceso, se genera una gran cantidad de informes y documentación que debe ser recopilada, estructurada y reportada de manera precisa.

Actualmente, parte de estos procesos se realiza manualmente, lo que supone un esfuerzo significativo por parte de los auditores. Estos procedimientos repetitivos no solo ralentizan el flujo de trabajo, sino que también aumenta la probabilidad de errores en la introducción de la información.

- *E-mail de contacto: marc.benitez.soler@gmail.com*
- *Mención realizada: Tecnologías de la Información*
- *Trabajo tutorizado por: Sergi Robles Martínez (Área de Ciencias de la Computación e Inteligencia Artificial)*
- *Curso 2024/25*

Realizar una auditoría puede llegar a ser un proceso largo y tedioso, donde algunos procesos cambian ligeramente de una auditoría a otra. El problema principal detectado es la falta te automatización en estos procesos repetitivos.

Dichos procesos constan de una serie de pasos que actualmente se realizan de manera manual, lo que puede suponer una serie de problemas:

- Tiempo excesivo en tareas repetitivas, desviando el enfoque de tareas más analíticas e importantes.
- Propensión a errores humanos.
- Baja eficiencia, generando mayor carga de trabajo y posibles retrasos.

Dado que una auditoría requiere de la realización de muchas tareas de este estilo, resulta fundamental optimizarlas mediante una solución informática que permita agilizar los procesos administrativos.

1.1 Objetivos

El objetivo principal detrás de este proyecto es desarrollar un aplicativo web funcional y eficiente que automatice tareas administrativas repetitivas asociadas al proceso de auditoría. Estas tareas, hasta ahora ejecutadas manualmente, suponen una carga operativa elevada y son propensas a errores humanos, lo cual afecta la eficiencia y fiabilidad del proceso.

Con el objetivo de optimizar este flujo de trabajo, el aplicativo permitirá al usuario cargar un archivo Excel previamente cumplimentado por el cliente y rellenar automáticamente un formulario web (ISA-315) con los datos extraídos del archivo.

A continuación, se define un listado específico de los objetivos del proyecto, con el nivel correspondiente de prioridad 1-3, siendo 3 el mayor nivel de prioridad.

1. Planificación del proyecto (3)
    a. Definir metodología y marco de trabajo
    b. Definir sprints y tareas
2. Diseño de la interfaz de usuario (2)
3. Implementación de la funcionalidad de carga y lectura de archivos Excel (3)
4. Proceso de datos extraídos para su posterior uso (3)
5. Creación e implementación del formulario que debe ser rellanado (2)
6. Automatización del rellenado del formulario en base a los datos procesados (3)
7. Realización de pruebas funcionales y validación del funcionamiento del aplicativo (1)
    a. Realizar posibles optimizaciones y/o mejoras
8. Documentación del proceso del desarrollo (1)

Este proyecto no solo busca mejorar la eficiencia en tareas administrativas dentro del proceso de auditoría, sino también servir como base para posibles futuras optimizaciones dentro de la empresa, fomentando una cultura de innovación y automatización.

2 METODOLOGÍA

Para el desarrollo de este proyecto se ha optado por seguir una metodología ágil, ya que se adapta perfectamente a las características del trabajo: un entorno cambiante, con necesidad de flexibilidad, entregas funcionales frecuentes y mejoras continuas.

Ágil [1] no es una metodología única, sino una familia de métodos que comparten los siguientes valores fundamentales a la hora de construir proyectos, valorando lo siguiente:

- A individuos y sus iteraciones frente a procesos y herramientas.
- Al software (producto) que funciona, que una documentación exhaustiva.
- La respuesta al cambio que el seguimiento de un plan.

En un proyecto de desarrollo de software o un aplicativo, una buena gestión no es suficiente y se necesita también seguir unas buenas prácticas de programación. Esto permitirá una gestión optimizada, a la vez que se crea un aplicativo de calidad. Es aquí donde entran en escena los métodos ágiles de programación, compartiendo los valores y principios ágiles.

En este proyecto nos basaremos en *Scrum* [2], un marco de trabajo que divide el proyecto en sprints, es decir, bloques cortos de tiempo en los que se planifica, desarrolla, prueba y revisa una parte concreta del aplicativo permitiendo así centrar el esfuerzo en objetivos específicos por fase. Utilizaremos *Scrum* de la siguiente manera:

- **Sprints**: Iteraciones donde se definen objetivos específicos
- **Backlog del producto**: lista priorizada de funcionalidades y tareas a realizar
- **Revisión y mejora continua:** Al final de cada sprint se ha realizado una pequeña evaluación personal para identificar aciertos, problemas y mejoras a implementar.

Combinaremos el marco de trabajo *Scrum* con una serie de prototipos en los que se implementará y testeará una serie de funcionalidades, realizando las fases necesarias requeridas en cada prototipo en específico (Análisis, diseño, desarrollo o testeo). El proyecto se dividirá en 3 fases principales, la primera donde se crea la interfaz de usuario y se manejan los datos extraídos del Excel, una segunda donde crearemos el formulario que debe ser rellenado y una última donde automatizaremos el proceso de rellenar el formulario con los datos previamente extraídos.

Debido a la naturaleza propia de este proyecto, donde uno de los requisitos más importantes es la flexibilidad, y donde se entrega un software funcional de forma rápida, iterativa y adaptativa, la metodología Ágil resulta idónea.

2.1 Planificación

Para planificar de forma clara y visual el desarrollo del aplicativo, se ha optado por combinar *Scrum* con un calendario del proyecto, lo que permite tener una visión del

tiempo del proyecto sin perder la flexibilidad propia de los métodos ágiles.

El calendario del proyecto refleja los tiempos estimados del proyecto indicando:

- Número de sprint correspondiente
- Tareas o funcionalidades clave por iteración
- Duración estimada de cada fase

CALENDARIO DEL PROYECTO

Sprint 1	Establecer metodología y planificación. Análisis de requisitos	17/03 – 6/04
Sprint 2	Análisis, diseño e implementación de la interfaz	07/04 – 19/04
Sprint 3	Implementación de la carga y lectura de Excel	20/04 – 02/05
Sprint 4	Análisis, diseño e implementación del formulario web.	03/05 – 14/05
Sprint 5	Procesamiento de los datos extraídos y automatización del rellenado de formulario	15/05 – 07/06
Sprint 6	Pruebas, corrección de errores y mejoras finales	08/06 – 19/06
Sprint 7	Documentación y cierre del proyecto	20/06 – 30/06

Fig. 1. Calendario del proyecto.

3 ESTADO DEL ARTE

En el ámbito de la automatización de tareas repetitivas, existen algunas herramientas y soluciones que abordan este problema desde diferentes enfoques. Como por ejemplo Microsoft Power Automate Desktop, que permite automatizar procesos mediante la creación de flujos de trabajo que interactúan con aplicaciones y servicios.

Aun así, requiere esa configuración y mantenimiento manual de los flujos, lo que puede ser poco escalable, además de no ser un proceso enteramente automático, no adaptándose completamente así a las necesidades de este proyecto.

3.1 JavaScript; NodeJS

JavaScript [4] [5] es el lenguaje de programación de la Web. La inmensa mayoría de las webs modernas utilizan JavaScript, y todos los buscadores web modernos (en escritorios, consolas, tablets y smartphones) incluyen intérpretes JavaScript, haciendo que JavaScript sea el lenguaje de programación más extenso de la historia. Todo programador web debe aprender este triad de tecnologías para desarrollo web: HTML para especificar el contenido de las páginas, CSS para especificar la presentación y JavaScript para especificar el comportamiento.

JavaScript es un lenguaje de programación de alto nivel y dinámico que se adapta perfectamente a estilos de programación orientada a objetos [6].

Aprovechando la ventaja del poder y la simplicidad de JavaScript, Node [3] transforma la difícil tarea de escribir aplicaciones del lado del servidor basadas en eventos en algo sencillo. Node proporciona una infraestructura no bloqueante puramente basada en eventos para crear software altamente concurrente y te permite construir fácilmente servicios de red rápidos y escalables.

Una de las características más potentes de Node [7] es que se basa en eventos y programación asíncrona. Imaginemos el caso donde un bloque de código contiene una línea en específico con una operación que consume mucho tiempo. Lo que pasaría en modelos de programación síncronos es que todas las líneas de código por debajo de la mencionada tendrían que esperar a que esta operación costosa acabase. Un modelo asíncrono maneja la situación de manera diferente, donde esta operación asíncrona es mandada a otro stack de ejecución, que resuelve la operación mientras el resto de código se ejecuta, y vuelve al stack de ejecución síncrono una vez ha terminado, a través de una callback que se ejecuta cuando el proceso ha finalizado.

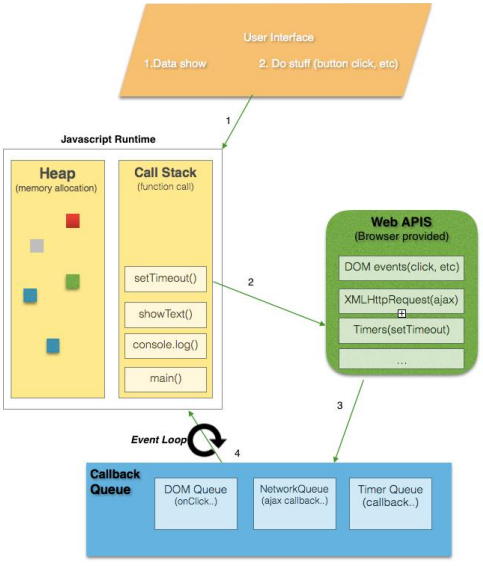


Fig. 2. Execution Stack de JavaScript.

3.2 Vue

En el desarrollo de este proyecto, se ha decidido incorporar Vue.js [11], un framework progresivo de JavaScript que ha ganado notable popularidad en los últimos años. Aunque es relativamente reciente en comparación con otros frameworks como React o Angular, Vue.js ha demostrado ser una herramienta poderosa y flexible para la construcción de interfaces de usuario dinámicas y eficientes.

Vue [8] [9] se caracteriza por su enfoque incremental, lo que significa que puede ser adoptado gradualmente en proyectos existentes. Su núcleo se centra en la capa de vista, facilitando la integración con otras bibliotecas o proyectos ya en marcha. Esta modularidad permite a los desarrolladores seleccionar las funcionalidades que necesitan, sin imponer una estructura rígida.

Una de las razones para elegir Vue.js en este proyecto es su sintaxis clara y sencilla, que facilita el aprendizaje y la implementación, especialmente para aquellos familiarizados con HTML, CSS y JavaScript. Además, Vue.js ofrece un sistema de componentes que permite dividir la interfaz en bloques reutilizables y autónomos, mejorando la mantenibilidad y escalabilidad del código.

Dado que el proyecto cuenta con una interfaz de usuario interactiva y reactiva, Vue.js se presenta como una opción adecuada, responde a una necesidad técnica manejando el estado de la aplicación de manera eficiente y es compatible con herramientas modernas de desarrollo.

La incorporación de Vue.js a este proyecto es una oportunidad de experimentar con un framework moderno que continúa ganando tracción en la comunidad de desarrollo web.

3.3 Creación de la aplicación

Antes de comenzar con el desarrollo del aplicativo, tenemos que crear la aplicación. Una vez tengamos instalado Node, creamos un directorio donde pretendemos crear el proyecto. Cuando estemos en el directorio creado, ejecutamos el siguiente comando en la línea de comandos:

```
$ npm create vue@latest
```

Este comando instalará y ejecutará *create-vue*, la herramienta oficial de creación de aplicaciones de Vue [10]. Al ejecutar el comando, se presentan una serie de preguntas en la terminal con la opción de añadir diferentes extras, en nuestro caso simplemente presionamos enter y no añadimos nada más.

Seguidamente, ejecutamos los siguientes comandos:

```
$ cd <nuestro-proyecto>; $ npm install; $ npm
run dev
```

Una vez hecho esto, ya tenemos nuestro proyecto creado y funcionando.

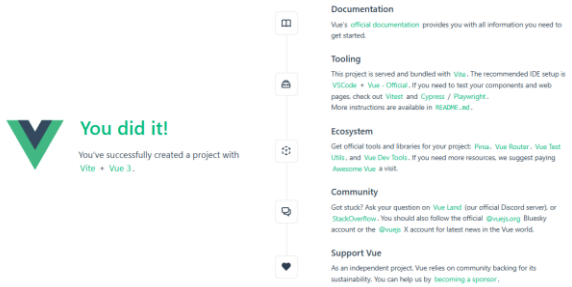


Fig. 3. App Vue una vez abrimos el navegador.

En la propia página, podemos explorar todos los ítems reactivos con las herramientas proporcionadas, útil si quieres familiarizarte con el entorno, en nuestro caso, eliminaremos todos los archivos y nos quedaremos con el esqueleto principal de un proyecto vacío, para implementar nuestro aplicativo, comenzando por la interfaz.

4 Análisis y requisitos

Este aplicativo se basa en una interfaz sencilla donde el usuario puede subir un archivo Excel, el cual contiene una serie de preguntas y respuestas ya contestadas por parte del cliente. Cada Excel hace referencia a un aplicativo que el cliente está usando y esa información sirve a los auditores para realizar las ISAs, un conjunto de reglas o estándares de actuación que se aplican en las auditorias de empresas con el fin de dar a los informes de auditoría resultantes uniformidad y validez a nivel internacional.

Una vez el usuario haya realizado esta sencilla acción, el aplicativo es capaz de extraer la información necesaria del Excel y rellenar el formulario específico de ISA-315 accediendo automáticamente. Esta acción, hasta ahora realizada manualmente, debe repetirse para cada aplicación que use cada cliente, por lo tanto, conseguir automatizarla reduce considerablemente el tiempo de ejecución y minimiza posibles errores humanos.

A continuación, se presenta el listado de requisitos funcionales y no funcionales que debe tener cada fase del aplicativo:

Interfaz de usuario

- RF1.1: Sistema permite subir un archivo Excel.
- RF1.2: El sistema valida que el archivo tenga la extensión y formato correctos.
- RF1.3: El sistema muestra un mensaje de confirmación cuando el archivo se cargue correctamente.
- RF1.4: El sistema no permite avanzar hasta que no se cargue el archivo.

Formulario web

- RF2.1: El sistema detecta los campos del formulario web creado.
- RF2.2: El sistema cuenta con todas las variantes de preguntas.

Automatización del proceso

- RF3.1: El sistema extrae las respuestas del archivo cargado.
- RF3.2: El sistema comprueba que el archivo no esté vacío y mapea correctamente los datos.
- RF3.3: El sistema rellena automáticamente los campos del formulario.
- RF3.4: El sistema permite visualizar los resultados.
- RF3.5: El sistema detecta errores durante el proceso y notifica de ellos.

Requisitos no funcionales

- RNF1: La interfaz será sencilla e intuitiva.
- RNF2: El tiempo de procesamiento y proceso de rellenado debe ser corto.
- RNF3: El sistema maneja errores de forma segura.
- RNF4: El sistema se desarrollará con las tecnologías mencionadas en el informe.
- RNF5: El sistema permite futuras mejoras.

5 Interfaz de usuario

En este apartado damos comienzo a la primera fase principal del proyecto, donde se diseña y se implementa la interfaz de usuario del aplicativo.

5.1 Análisis

La interfaz de usuario debe ser lo más sencilla, directa y funcional posible, permitiendo al usuario completar el proceso con un mínimo de interacción.

Siguiendo los requisitos descritos, la interfaz debe permitir al usuario subir un archivo y proporcionar una URL. Se validará que el archivo subido sea un Excel.

La interfaz permite avanzar y activa el botón de rellenar el formulario únicamente cuando ambos pasos se hayan realizado correctamente, reduciendo así errores y mejorando la experiencia de uso.

5.2 Diseño

Se ha diseñado una interfaz minimalista y orientada a la usabilidad, que busca priorizar la eficiencia de la interacción.

El diseño propuesto se divide visualmente en cuatro secciones alineadas verticalmente:

1. Encabezado
2. Zona de carga del archivo
3. Campo de texto de entrada de la URL
4. Botón de ejecución del proceso

Seguiremos una distribución vertical, centrada en la página y con espaciado claro entre secciones. En cuanto a la estética seguiremos una paleta de colores neutros y una topología legible y moderna junto a iconos simples para representar la carga de archivos, el cuadro de texto y el botón final.

Fig. 4. Mockup de la interfaz.

Por último, con la intención de perfeccionar la experiencia de usuario, implementaremos textos alternativos junto a los iconos, y mensajes claros de error y éxito visibles y comprensibles.

5.3 Implementación

Una vez finalizado el proceso de análisis y diseño, se inicia la implementación de la interfaz. El objetivo de esta fase es transformar los requisitos funcionales definidos en la fase previa en algo real y funcional.

La aplicación se ha desarrollado siguiendo una estructura de componentes, donde cada componente representa una sección funcional de la interfaz y ciertos componentes pueden contener otros componentes como hijos. La aplicación se ha dividido en dos componentes principales, de los cuales surgen los demás.

App.vue: Componente raíz de la aplicación, desde donde se importa y se utiliza la estructura principal, los dos componentes principales Interfaz.vue y Formulario.vue. También almacena el archivo Excel que sube el usuario y la url que introduce en caso de necesitarla, estos datos los recibe a través de emisiones que realizan los componentes principales. El componente Interfaz se muestra únicamente si la variable showForm es falsa a través del parámetro Vue `v-if="!this.showForm"`, en cualquier otro caso (v-else en el componente Formulario) se mostrará el componente Formulario.

El componente Interfaz también cuenta con el parámetro `@submit="handleSubmit"`, el cual activa dicho método, que recibe dos parámetros, el libro Excel subido y la url introducida, para seguidamente almacenarlos en las variables excel y url. También cambia el valor de showForm a true, para que se muestre el formulario.

Componente hijo de App.vue

Interfaz.vue: Gestiona la subida del archivo a través del evento fileLoaded recibido desde Archivo.vue, la URL introducida a través del evento urlUpdated recibida desde URL.vue y el evento del click al botón de rellenar el formulario a través del evento submit recibido desde BotonForm.vue.

Cuando fileLoaded se activa, se ejecuta el método onFileLoaded, el cual recibe el parámetro workbook, que cambia el valor de la variable this.fileLoaded a true y almacena el workbook recibido por parámetro en la variable this.excel. Este procedimiento es el mismo que sigue el método onUrlUpdated cuando se activa urlUpdated, guardando el valor de la url recibida por parámetro en la variable local.

Por otro lado, cuando el evento submit se activa, simplemente emite el evento submit de nuevo, pasando por parámetro al componente App.vue el excel y la url almacenadas anteriormente.

También activa o desactiva el botón de rellenar el formulario a través del prop `:disabled="!fileLoaded || !currentUrl"`, el cual comprueba que se haya introducido una url y que se haya subido un archivo.

Componentes hijos de Interfaz.vue

Cabecera.vue: Componente únicamente visual, contiene el título principal y un subtítulo, ambos como props recibidos desde el padre.

Archivo.vue: Gestiona toda la lógica de subida del archivo, contiene un elemento HTML `input type="file"` donde el usuario puede subir el Excel, comprobando la extensión con `accept=".xlsx, .xls"`. Desde este componente se ejecuta el método onFileChange a través del parámetro @change cada vez que se realiza un cambio en el input, emitiendo el archivo recibido al padre a través de `this.$emit('fileLoaded', file)`.

URL.vue: Contiene un elemento HTML `input type="text"` con el atributo Vue `v-model="formUrl"`, lo que nos permite almacenar en la variable local formUrl de forma automática el texto que el usuario introduzca en el cuadro de texto, también cuenta con el atributo `@keydown.enter.prevent` para evitar que al pulsar intro realice cualquier acción, ya que queremos que únicamente actúe el botón de rellenar el formulario. El input contiene también un `@input="onUrlChange"`, que ejecuta dicho método cada vez que el usuario introduce un texto, el método emite el evento `this.$emit("urlUpdated", this.formUrl)`, pasándole el texto introducido a App.vue.

BotonForm.vue: Gestiona el evento de submit una vez el usuario ya ha completado todos los pasos anteriores, contiene un elemento HTML button, el cual se activa o desactiva a través del prop disabled que recibe de App.vue. Controlamos el evento submit a través de `@click="this.$emit("submit")"`, que emite dicho evento cuando se hace click en el botón.

Todos estos componentes se juntan e interactúan entre ellos creando así la interfaz de usuario, rápida y reactiva, ofreciendo feedback inmediato a cada acción del usuario.

6 FORMULARIO WEB

Después de finalizar la primera fase del proyecto donde completamos la interfaz de usuario, en este apartado comenzamos la segunda fase, adentrándonos en la creación de los formularios y toda la lógica detrás de estos.

6.1 Análisis

El formulario es el destino de los datos cargados desde el archivo Excel. Este componente representa un cambio de vista en la aplicación, ya que sustituirá a la interfaz inicial una vez el usuario presione el botón de rellenar el formulario.

El formulario que se ha creado es una copia que simula un entorno de trabajo real, donde el usuario debe rellenar el formulario a través de los datos en el Excel. Este enfoque nos permite realizar un mapeo directo entre los datos extraídos y los campos del formulario, teniendo un control total del comportamiento dinámico de este.

6.2 Diseño

El formulario seguirá una estructura clara y compatible con el proceso de relleno automático que se llevará a cabo en la fase final del proyecto.

Este componente no es accesible inicialmente, sino que se genera de manera dinámica tras la acción del usuario sobre el botón de rellenar el formulario. En ese momento la pantalla principal cambia, mostrando el formulario rellenado automáticamente con los datos extraídos.

La estructura visual del formulario está diseñada de manera vertical y dividida en bloques, los cuales representan grupos de preguntas basadas en distintos temas, todas sobre la misma aplicación. Así pues, tendremos un formulario dividido en formularios "más pequeños". Cada bloque contará con los siguientes elementos:

- Título correspondiente a la sección que estamos tratando (Ajustes de seguridad, Cambios…)
- Área de texto o input correspondiente a cada pregunta y respuesta.

Aunque el objetivo principal sea que el formulario se rellene de forma automática, se podrá revisar el contenido, modificar alguna respuesta en las áreas de texto o realizar correcciones oportunas.

6.3 Implementación

El objetivo es construir un componente que actúe como un formulario web, el cual será rellenado automáticamente, siendo destino de los datos extraídos del archivo Excel después de haberlos recibido. Este componente se genera de forma dinámica tras la interacción del usuario con el botón de "Rellenar formulario", sustituyendo a la interfaz inicial por el formulario general.

Este formulario se divide a su vez en distintos formularios más pequeños, uno por cada tema. Cada formulario contiene un grupo de preguntas específicas y los campos de respuesta correspondientes, que serán rellenados automáticamente con la posibilidad de ser modificados por el usuario. A continuación, se describen todos los componentes que forman esta parte del aplicativo y todas las interacciones entre ellos que permiten su control y uso.

App.vue: Como hemos visto anteriormente, el componente raíz de App.vue contiene los dos componentes principales y los datos que ha introducido el usuario a través del componente Interfaz.vue. Este componente mostrará el componente hijo Fomulario cuando la variable showForm sea true, y cuando hayamos recibido los datos del servidor.

Una vez el usuario haya clickado en el botón de Rellenar formulario, se mostrará una sencilla pantalla de carga mientras esperamos los datos del servidor, una vez los hayamos recibido, automáticamente mostrará el formulario.

Componente hijo de App.vue

Formulario.vue: Este componente muestra los diferentes formularios y gestiona toda la lógica de las respuestas para que el usuario pueda modificar cualquier campo o respuesta. Se trata de un formulario reactivo donde existen preguntas fijas y otras que se muestran dependiendo de respuestas que se hayan introducido anteriormente.

Cada formulario está contenido en un componente HTML form, el cual contiene un título (h2), una serie de bloques contenidos en un div, que contienen la pregunta

que corresponda (label) y un componente Respuesta, el cual describiremos más adelante. De esta manera, podemos gestionar todas las preguntas y respuestas de manera sencilla y limpia. Para almacenar todas las respuestas, se ha creado una lista para cada formulario (respuestasA[ ], respuestasB[ ]…), la cual contendrá todas las respuestas del formulario en orden ascendiente.

Cada componente Respuesta recibe el prop `type="text"` o `type="select"` (que determina qué tipo de respuesta requiere la pregunta, ya sea un cuadro de texto de respuesta abierta o una selección de Sí/No), y el prop modelValue, el cual se recibe automáticamente al utilizar el atributo `v-model="respuestasX[i]"` (donde X es el formulario e i es el índice de cada respuesta), que enlaza directamente el prop a la variable local, la cual se modificará a través del evento update recibido por el hijo.

Componentes hijos de Formulario.vue

Cabecera.vue: Es el mismo componente que se utiliza en el archivo Interfaz.vue, se puede reutilizar cambiando los valores de los props del título y subtítulo.

Respuesta.vue: Como cada pregunta iba seguida de una respuesta, el código podía llegar a ser muy repetitivo, confuso y poco eficiente. Esto se ha solucionado creando un componente que gestiona qué tipo de respuesta se muestra y donde se almacena la respuesta introducida.

El componente contiene dos bloques de respuesta diferentes, el primero es un `input type="text"` que se muestra cuando el prop recibido es igual a "text" a través del atributo Vue `v-if="type==='text'"`. Tiene también los atributos `@keydown.enter.prevent` (que impide que la tecla Enter tenga algún efecto), `:value="modelValue"`, que enlaza el prop recibido por el v-model de la respuesta correspondiente al valor value que se muestra en el cuadro de texto y por ultimo:

```
@input="$emit('update:modelValue',$event.tar-
                get.value)"
```

que actualiza el valor de la respuesta en caso de que exista cualquier modificación por parte del usuario. El segundo bloque de respuesta es un tag div, que se muestra únicamente si el prop recibido es igual a "select" a través del atributo Vue `v-if="type === 'select'"`. Dicho div contiene a su vez dos `input type="radio"`, con los atributos `:checked="modelValue==='SI/NO'"` y `@change="seleccionar('SI/NO')` para asignar dinámicamente la respuesta seleccionada por el usuario con la respuesta pasada como prop, a través de la función `seleccionar()`, que emite el valor al padre con el valor pasado por parámetro.

Estos últimos inputs, contienen también el atributo `@click="onClick(true/false)"` que se implementa para dejar en null una respuesta seleccionada si vuelves a pulsar sobre un botón previamente seleccionado.

7 AUTOMATIZACIÓN DEL PROCESO

Una vez finalizado el desarrollo de los dos componentes visuales del proyecto, toca adentrarse en la parte lógica, donde se verá la implementación realizada para automatizar el proceso de relleno del formulario a través del archivo Excel subido.

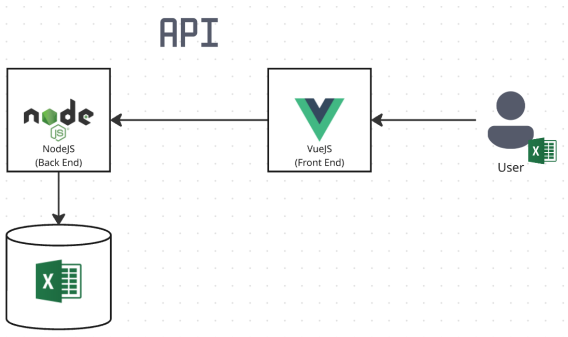


Fig. 5. Diagrama visual del funcionamiento del aplicativo.

En esta imagen podemos observar el funcionamiento global del aplicativo.

A continuación, veremos las implementaciones y modificaciones realizadas siguiendo una serie de pasos.

7.1 Frontend (Envío del archivo)

Desde el Frontend se envía el archivo que el usuario ha subido y que internamente se ha almacenado en App.vue. El envío se hace cuando se ejecuta la función `handleSubmit()` al hacer click en el botón de "Rellenar formulario", ejecutando las siguientes líneas de código:

```
const formData = new FormData();
formData.append('file', this.excel);

axios.post('http://localhost:3000/pro-
cess_form', formData)
.then(res => {
  console.log('Datos recibidos: ', res.data);
  this.data = res.data;
})
.catch(err => {
  console.error('Error   procesando   Excel:',
err);}
);
```

donde podemos observar que se crea un objeto FormData, al cual añadimos el archivo subido, para posteriormente enviarlo al endpoint del servidor a través de una peticion POST con Axios.

Si la respuesta del servidor con los datos ya mapeados y extraídos es correcta, la promise de la petición resuelve con los datos recibidos y los almacenamos en una variable local para poder trabajar con ellos. Por otra parte, si la promise rechaza, se mostrará el error recibido en la consola.

7.2 Backend (Extracción y mapeo de datos)

Tras recibir el archivo que el usuario ha subido, montar el servidor y crear el endpoint correspondiente, el servidor se encarga de guardar el archivo en memoria, leerlo, extraer los datos que queramos y almacenarlos en un diccionario con todas las respuestas que necesitemos para posteriormente enviarlo de nuevo al Frontend, que se encargará de mostrarlo.

A continuación, se explica el código (simplificado) del Backend paso por paso:

1. Creación del servidor usando Express.js y de variables necesarias.

```
upload=multer({storage:      multer.memoryStor-
age()});
app = express();
port = 3000;
app.use(cors());

data = {};
respuestasA = Array(8).fill('');
…
```

2. Creamos el endpoint al que se accederá desde el Frontend, especificamos que recibiremos un archivo y el proceso no continua si no lo recibimos.

```
app.post('/process_form', file), (req,res) =>{

if (!req.file) {
  console.error();
  return res.status(400);
}
```

3. Se utiliza un bloque try/catch para tratar de ejecutar la sección de código, evaluando cualquier posible error, leemos el archivo subido, y convertimos la hoja de la cual queremos leer los datos en un JSON [12].

```
  try{
    excel = XLSX.read(file, {type: 'buffer'});
    hoja = excel.Sheets['Hoja1'];
    filas = XLSX.utils.sheet_to_json(hoja);
```

4. Primero, almacenamos todas las respuestas del Excel en un diccionario data, mapeando así todas las respuestas y datos, cada celda del Excel está almacenada en la posición correspondiente del array (la posición [2][2] equivale a la celda 3C).

```
  data = {
    info: {
      nombre_empresa: filas[2][2]
      …
    },
    general: {
      respuesta1: filas[12][5]
      …
    }
    servicios: {
      respuesta1: filas[15][5]
      …
    } // Y así para cada sección
} // data
```

5. Después de mapear todos los datos que necesitamos, los guardamos en los arrays de respuestas, que imitan a los arrays del Frontend, con las respuestas ordenadas por el orden del formulario final, una vez hecho esto, guardamos todos los arrays en un diccionario, que enviamos al frontend en formato JSON.

```
if (data.cambios.respuesta1 === 'SI'){
  respuestasA[1]=data.cambios.respuesta2;
  respuestasA[2]=data.cambios.respuesta3;
  respuestasA[3]=data.cambios.respuesta6;
  …
}

const final_data = {
  respuestasA,
  respuestasB,
  respuestasC,
  respuestasD,
  respuestasE
}

res.json(final_data);
}
```

6. Por último, gestionamos cualquier error de ejecución que haya podido ocurrir con el catch.

```
catch(error){
  console.log('Error:', error); }
```

7. Activamos el servidor y esperamos a la respuesta del cliente con el archivo correspondiente.

```
app.listen(port, () => {
    console.log(`Example app listening on port
${port}`)
})
```

7.3 Consumo del JSON y mostrar forms

Después de terminar toda la lógica del backend y enviar los datos al cliente, falta mostrar los datos en los formularios correspondientes.

Desde el Frontend, una vez recibimos los datos y los guardamos en la variable local como hemos visto en los apartados anteriores, le pasamos estos datos recibidos como prop a Formulario.vue.

```
<Formulario v-else :recieved_data="data"/>
```

Después, en el formulario, creamos un watcher del prop recieved_data(data), que está atento a cualquier cambio que reciba el prop, para ejecutarse en el momento en el que éste recibe un cambio (cuando recibimos los datos del servidor) para almacenar en los arrays de respuestas locales, los arrays de respuestas que hemos recibido (`this.respuestasX = data.respuestasX`).

Una vez almacenamos los resultados, se muestran automáticamente por pantalla, en el bloque de respuesta que corresponde, gracias al atributo Vue v-model que enlaza bidireccionalmente las respuestas que tenemos con las respuestas del usuario, el usuario finalmente puede revisar el contenido de las respuestas, y modificar lo que considere necesario.

8 Resultados

Una vez finalizado el desarrollo del aplicativo, se han llevado a cabo dos pruebas funcionales con el objetivo de evaluar el rendimiento lógico y estructural del aplicativo.

En la primera prueba, se ha simulado el proceso completo de rellenar los formularios correspondientes a partir de un archivo Excel previamente completado, realizando dos versiones del mismo procedimiento:

- Prueba manual, donde el usuario ha rellenado los formularios manualmente extrayendo los datos de las respuestas obtenidas.
- Prueba de rellenado automático de los formularios utilizando la aplicación.

Esta prueba manual se ha realizado un total de 5 veces, donde el tiempo total invertido (de media) en rellenar los formularios ha sido de 9 minutos 15 segundos, mientras que, en la prueba automática, el tiempo invertido se reduce al tiempo que se tarda en revisar las respuestas y realizar correcciones si se necesita, aproximadamente 2 minutos de media.

Dicho esto, podemos concluir que la diferencia en el proceso de rellenar los formularios es notablemente significativa, con una mejora de entre el 70 - 80% en el tiempo de ejecución, además de suprimir posibles errores humanos que se cometan durante el proceso.

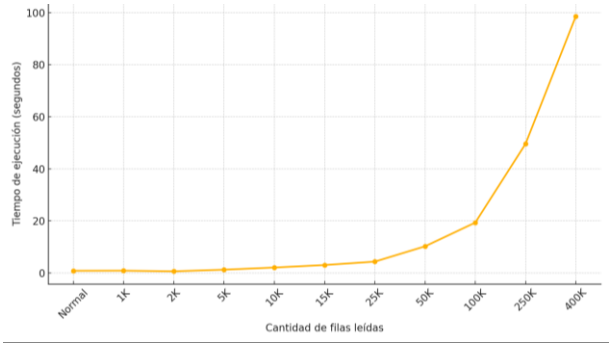


Fig. 6. Grafico:
Tiempo de ejecución al ejecutar diferentes archivos.

Para la segunda prueba, se ha realizado una prueba de resistencia del aplicativo, aumentando el número de filas en cada ejecución, para evaluar el comportamiento del tiempo de ejecución.

Podemos observar que el aplicativo funciona de forma idónea hasta las 25.000 respuestas, a partir de ahí, se observa una degradación progresiva del rendimiento, aumentando el tiempo de ejecución exponencialmente a partir de las 100.000 respuestas, llegando al fallo del sistema en medio millón de respuestas leídas.

9 Conclusiones

A lo largo de este proyecto se ha conseguido implementar con éxito un aplicativo que cumple con el objetivo principal de automatizar tareas administrativas, a través de la automatización de rellenado de formularios con los datos extraídos de un archivo Excel subido por el usuario. Esta implementación nos ha permitido observar cómo efectivamente, hemos acertado con la teoría inicial de mejorar drásticamente el tiempo empleado en la tarea, haciendo el proceso mucho más dinámico y eficaz, reduciendo también posibles errores.

No obstante, se han sufrido ciertas complicaciones durante el transcurso del proyecto, tales como no poder utilizar el archivo Excel de trabajo original, o no poder trabajar en el entorno web de la empresa, por lo que se han tenido que llevar a cabo modificaciones, y trabajar con copias, tanto del archivo como de los formularios, para simular el entorno de trabajo real. Estas modificaciones han significado no poder completar el aplicativo introduciendo la URL donde se encuentra el formulario original.

Este proyecto cumple una función práctica inmediata, que podría mejorarse implementando una base de datos, añadiendo más compatibilidad con otro tipo de archivos, o implementando funcionalidades más avanzadas. El proyecto constituye una base sobre la cual se puede trabajar, para adaptarla a cualquier entorno de trabajo o necesidad.

Agradecimientos

Quiero aprovechar para expresar mi más sincero agradecimiento a todas las personas y entidades que me han ayudado en este proceso.

Gracias a mi tutor Sergi Robles por todo el apoyo, correcciones y ayuda que han servido para orientarme en este proyecto.
Gracias a EY por permitirme trabajar con ellos, ofreciendo un espacio donde poder aprender y trabajar en este proyecto.
Por último, este proyecto está dedicado a Susana, Antonio y Daniel, mi familia, y a mis amigos, mis dos pilares fundamentales, que me han estado ayudando y apoyando en todas y cada una de las fases de mi vida, sin vosotros no sería quien soy y todo esto no sería posible.

BIBLIOGRAFÍA

[1] A. Cockburn. Agile and Self-Adapting. *Agile Software Development.* Highsmith Series Editors. pp. 145-162. 2000.

[2] C. Lasa, A. Álvarez y R. de las Heras. Métodos Ágiles. *Métodos Ágiles Scrum, Kanban, Lean*. Anaya Multimedia. pp. 24-39. 2018.

[3] L. Orsini. *What You Need to Know About Node.js*. read write. [Online]. Disponible en: http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/. 2013. [Consultado 10 abril 2025].

[4] D. Flanagan. Introduction to JavaScript. *JavaScript: the definitive guide*. O'Reilly Media, Inc. pp 1-4. 2011.

[5] P. Teixeira. Introducing Node. *Building JavaScript-Based Scalable Software.* John Wiley & Sons, Inc. pp 15-20. 2012

[6] M. Satheesh, B. J. D'mello y J. Krol. Welcome to JavaScript in the Full Stack. *Web Development with MongoDB and NodeJS.* Packt Publishing. Pp 1-12. 2015

[7] H. Shah, T.R. Soomro. *NodeJS challenges in implementation*. Global Journal of Computer Science and Technology. *17*(2). 73-83. 2017

[8] *¿Para qué se utiliza Vue.js?* RootStack. [Online]. Disponible en: https://rootstack.com/es/blog/para-que-se-utiliza-vuejs?. 2022. [Consultado 17 abril 2025].

[9] VueJS.org. *Introducción. ¿Qué es Vue.js?* [Online]. Disponible en: https://es.vuejs.org/v2/guide/. [Consultado 17 abril 2025].

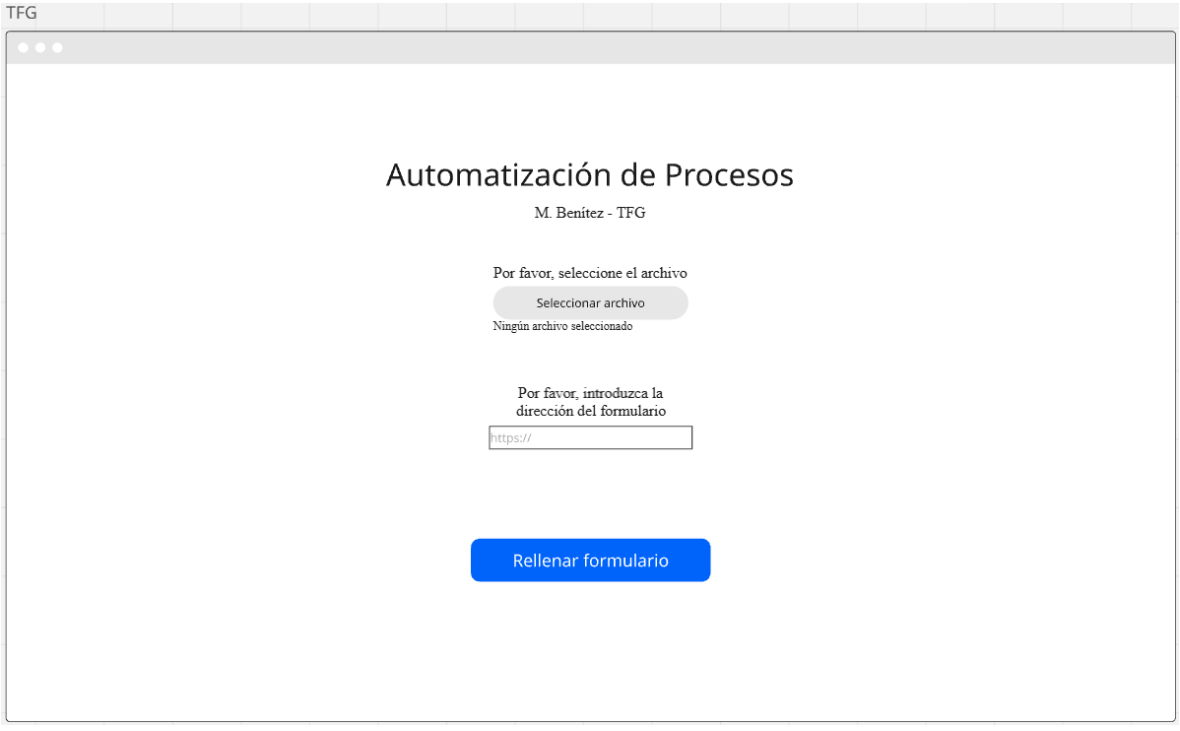
[10] VueJS.org. *Quick Start*. [Online]. Disponible en: https://vuejs.org/guide/quick-start.html. [Consultado 17 abril 2025].

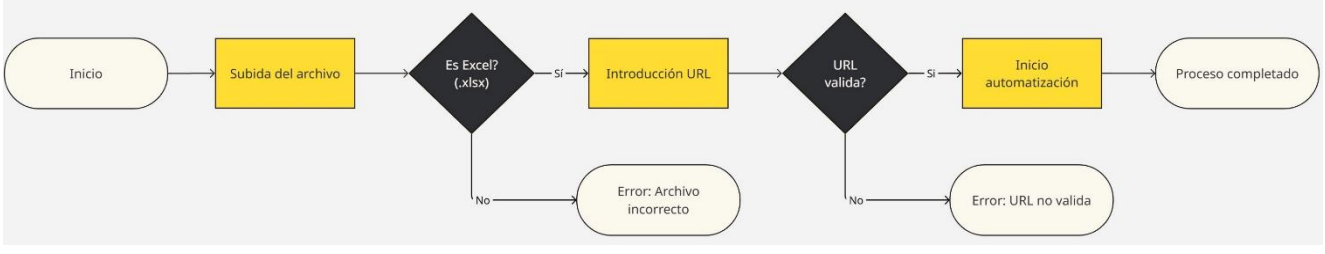
[11] J.A. Dongil. *Por qué elegir VueJS: 5 razones para considerarlo nuestro próximo framework de referencia.* GenBeta. [Online]. Disponible en: https://www.genbeta.com/desarrollo/por-que-elegir-vuejs-5-razones-para-considerarlo-nuestro-proximo-framework-de-referencia. [Consultado 25 Abril 2025

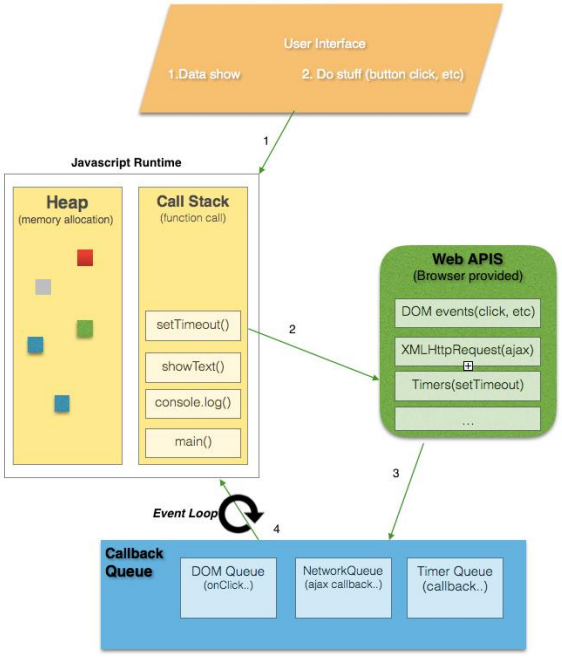
[12] GeeksForGeeks. *How to Read and Write Excel file in Node.js?*. [Online]. Disponible en: https://www.geeksforgeeks.org/how-to-read-and-write-excel-file-in-node-js/. [Consultado 10 Junio 2025].

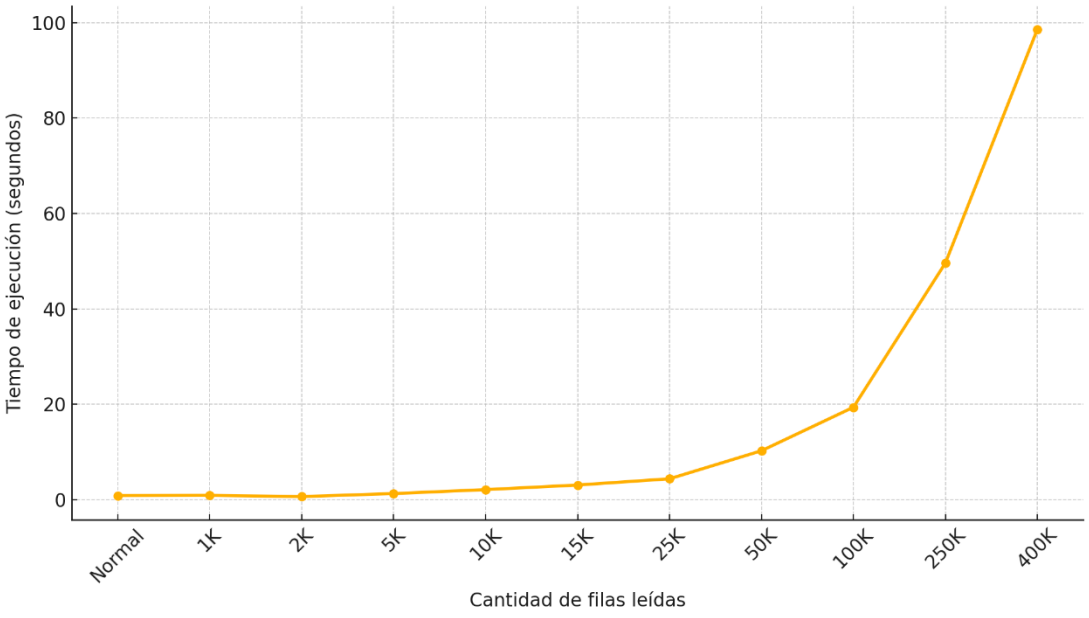
APÉNDICE

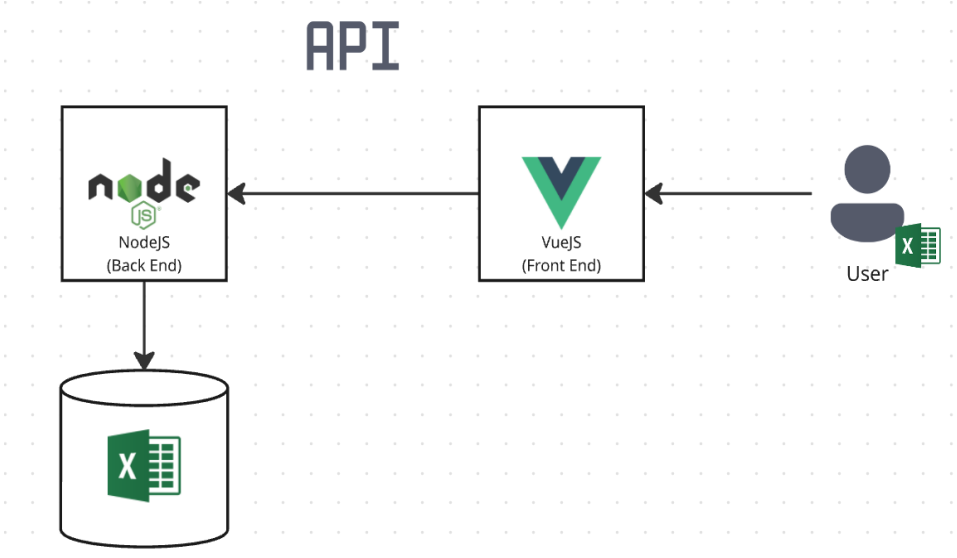
A1. IMÁGENES








Tiempo de ejecución (segundos)
0
20
40
60
80
100
Normal
1K
2K
5K
10K
15K
25K
50K
100K
250K
400K
Cantidad de filas leídas


API
NodeJS
(Back End)
VueJS
(Front End)
User