UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Desarrollo de APP para la conversión de EXCEL a procesos en la plataforma EGAM

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Marín Llorens, Alejandro

Tutor/a: Guerola Navarro, Vicente

CURSO ACADÉMICO: 2023/2024

Resumen

El presente trabajo de fin de grado aborda el desarrollo de una aplicación web destinada a la conversión eficiente de datos desde archivos Excel hacia la plataforma interna de la empresa EGAM. Esta iniciativa surge para optimizar y agilizar los procesos de transferencia de datos dentro de la organización, mejorando la eficiencia operativa y la gestión de información.

El objetivo principal del proyecto es diseñar e implementar una interfaz web intuitiva y funcional que facilite la interacción del usuario y asegure una conversión precisa y rápida de los datos desde Excel hacia los procesos de la plataforma EGAM. Para alcanzar este objetivo, se han utilizado tecnologías web modernas como C#, JavaScript (con la librería JointJS para la visualización de diagramas), y HTML.

El desarrollo se centra en la creación de funcionalidades que permitan mostrar los datos de Excel en forma de tabla, así como la visualización gráfica de estos datos mediante diagramas de flujo y diagramas de tortuga. La integración de estas herramientas proporciona una solución integral para la gestión y transformación de datos complejos en la empresa, facilitando la toma de decisiones y mejorando la eficiencia operativa.

La aplicación resultante no solo cumple con los requisitos técnicos y funcionales establecidos, sino que también proporciona una plataforma escalable y robusta que se alinea con las necesidades actuales y futuras de EGAM en términos de manejo de datos y procesos internos.

Palabras clave:

- Aplicación Web
- Conversión de datos
- Excel
- EGAM
- C#
- JavaScript
- HTML

Resum

El present treball de fi de grau aborda el desenvolupament d'una aplicació web destinada a la conversió eficient de dades des d'arxius Excel cap a la plataforma interna de l'empresa *EGAM. Aquesta iniciativa sorgeix per a optimitzar i agilitzar els processos de transferència de dades dins de l'organització, millorant l'eficiència operativa i la gestió d'informació.

L'objectiu principal del projecte és dissenyar i implementar una interfície web intuïtiva i funcional que facilite la interacció de l'usuari i assegure una conversió precisa i ràpida de les dades des d'Excel cap als processos de la plataforma *EGAM. Per a aconseguir aquest objectiu, s'han utilitzat tecnologies web modernes com a C#, JavaScript (amb la llibreria *JointJS per a la visualització de diagrames), i HTML.

El desenvolupament se centra en la creació de funcionalitats que permeten mostrar les dades d'Excel en forma de taula, així com la visualització gràfica d'aquestes dades mitjançant diagrames de flux i diagrames de tortuga. La integració d'aquestes eines proporciona una solució integral per a la gestió i transformació de dades complexes en l'empresa, facilitant la presa de decisions i millorant l'eficiència operativa.

L'aplicació resultant no sols compleix amb els requisits tècnics i funcionals establits, sinó que també proporciona una plataforma escalable i robusta que s'alinea amb les necessitats actuals i futures de *EGAM en termes de maneig de dades i processos interns.

Paraules clau:

- Aplicació Web
- Conversió de dades
- Excel
- EGAM
- C#
- JavaScript
- HTML

Abstract

This final degree project deals with the development of a web application for the efficient conversion of data from Excel files to the internal platform of the company EGAM. This initiative arises to optimize and streamline data transfer processes within the organization, improving operational efficiency and information management.

The main objective of the project is to design and implement an intuitive and functional web interface that facilitates user interaction and ensures an accurate and fast conversion of data from Excel to the EGAM platform processes. To achieve this goal, modern web technologies such as C#, JavaScript (with the JointJS library for diagram visualization), and HTML have been used.

The development focuses on the creation of functionalities that allow the display of Excel data in tabular form, as well as the graphical visualization of this data by means of flowcharts and turtle diagrams. The integration of these tools provides a comprehensive solution for the management and transformation of complex data in the company, facilitating decision making and improving operational efficiency.

The resulting application not only meets the established technical and functional requirements, but also provides a scalable and robust platform that aligns with EGAM's current and future needs in terms of data management and internal processes.

Keywords:

- Web application
- Data conversion
- Excel
- EGAM
- C#
- JavaScript
- HTML

Índice

Indice

1. Introducción

Este trabajo de final de grado (TFG) trata de una colaboración con una start-up de desarrollo de software enfocada en la digitalización de procesos con el fin de llevar el bienestar y tranquilidad a las empresas. Trataremos de desarrollar una aplicación web para la digitalización de procesos.

1.1. Motivación

La motivación para embarcarme en este proyecto de fin de grado surge de mi experiencia durante las prácticas en empresa. Durante este periodo, tuve la oportunidad de presenciar de cerca los desafíos que las empresas enfrentan al intentar implementar soluciones tecnológicas complejas, que a menudo resultan ser costosas y difíciles de gestionar. Esta experiencia me proporcionó una comprensión profunda de las barreras y los obstáculos que surgen en la integración de tecnologías avanzadas dentro del entorno empresarial. Asimismo, me permitió apreciar la importancia crucial de desarrollar soluciones eficientes y económicas que realmente respondan a las necesidades y limitaciones de las organizaciones. Por tanto, mi objetivo con este trabajo es explorar cómo podemos superar estos desafíos mediante estrategias innovadoras y prácticas, con el fin de facilitar la adopción efectiva de tecnologías que impulsen el crecimiento y la competitividad empresarial.

1.2. Objetivos

El desarrollo de la aplicación web tiene como objetivos principales los siguientes:

- **Transformación de datos a JSON**: Uno de los objetivos clave es facilitar la transformación de datos en formato JSON. La aplicación web permitirá convertir datos de diferentes fuentes o formatos en JSON de manera eficiente y precisa.
- **Fácil lectura de un proceso**: Otro objetivo importante es mejorar la legibilidad y comprensión de los procesos dentro de la empresa. La aplicación proporcionará herramientas visuales y descriptivas que permitirán a los usuarios identificar rápidamente el estado y el progreso de cualquier proceso digitalizado.
- **Generar un proceso digitalizado en el software de la empresa**: La aplicación web se centrará en facilitar la digitalización de procesos dentro del entorno de la empresa. La meta es proporcionar herramientas que permitan crear, gestionar y monitorear procesos digitalizados de manera eficiente y efectiva, mejorando así la productividad y la precisión en la ejecución de tareas.

1.3. Habilidades desarrolladas durante el TFG

Durante la realización de este trabajo, he consolidado habilidades previamente adquiridas durante mi grado, como el dominio de los lenguajes de programación C# y JavaScript, fundamentales para el desarrollo de aplicaciones web modernas. He mejorado significativamente mi capacidad para escribir código eficiente en ambos lenguajes, comprender los principios de la programación orientada a objetos en el caso de C#, y manejar de manera efectiva el DOM y eventos en JavaScript.

Además, he ampliado mis conocimientos al diseñar, desarrollar y desplegar aplicaciones web utilizando tecnologías que no habían sido abordadas en el plan de estudios. Esto me ha permitido explorar nuevas herramientas y frameworks que son relevantes en el ámbito profesional actual.

Otro aspecto fundamental ha sido la habilidad para planificar y organizar proyectos completos de desarrollo de software, desde la concepción inicial hasta la implementación final. Esta experiencia no solo ha fortalecido mi capacidad para gestionar eficazmente recursos y plazos, sino que también me ha proporcionado una comprensión más profunda de las metodologías y prácticas recomendadas en la industria del desarrollo de software.

Para concluir, este trabajo de fin de grado ha sido una oportunidad invaluable para aplicar y perfeccionar mis habilidades existentes, así como para adquirir nuevos conocimientos prácticos que son esenciales para mi futura carrera profesional en ingeniería informática y desarrollo de software.

1.4. Estructura de la memoria

La estructura de esta memoria comenzará por la contextualización del proyecto con un análisis del estado del arte y una breve muestra de las tecnologías utilizadas durante el proyecto. Además, se hablará de sus principales funcionalidades y otros proyectos similares al mismo.

A continuación, se abordarán los fundamentos del proyecto, incluyendo términos clave que ayudarán a comprender mejor el trabajo, ya que se repetirán a lo largo de la memoria.

Seguidamente se hablará de la metodología y los pasos seguidos para la realización de este proyecto, desde su inició hasta su final.

Cuarto, una breve introducción a los archivos que componen este programa antes de profundizar en cada uno de ellos explicando todos sus componentes y como interactúan con el código, así como su representación en el código.

Después mostraremos con un ejemplo los resultados obtenidos al finalizar este proyecto con un breve caso de uso de la aplicación misma.

Antes de concluir, se discutirán posibles mejoras que podrían realizarse en el futuro para optimizar la aplicación.

Finalmente, se incluirá un capítulo con la bibliografía utilizada y el anexo del proyecto.

2. Estado del Arte

2.1. Introducción

En el contexto actual de la transformación digital, la visualización efectiva de datos y la gestión eficiente de flujos de trabajo son fundamentales para mejorar la productividad y optimizar la toma de decisiones en las organizaciones. Este estado del arte examina un proyecto específico que emplea tecnologías avanzadas para lograr una visualización dinámica e interactiva de datos empresariales.

2.2. Tecnologías Utilizadas

El proyecto se fundamenta en el uso de un conjunto diverso de tecnologías frontend y herramientas especializadas que son cruciales para la representación visual y la manipulación efectiva de datos complejos:

- **Semantic UI, HTML y CSS**: Estas tecnologías son fundamentales para proporcionar la estructura y el estilo necesarios en la interfaz de usuario. Semantic UI ofrece un conjunto de componentes semánticos que facilitan la creación de interfaces modernas y adaptables. HTML y CSS, por su parte, permiten definir la estructura y el diseño visual de la aplicación web, asegurando una experiencia de usuario intuitiva y atractiva. La combinación de estas tecnologías asegura que la interfaz sea consistente, fácil de navegar y estéticamente agradable.

Imagen Semantics Ui

- **jQuery y AJAX**: Estas herramientas son esenciales para la interacción dinámica con el servidor. jQuery simplifica la manipulación del DOM y el manejo de eventos en JavaScript, mientras que AJAX permite realizar peticiones asíncronas al servidor sin necesidad de recargar la página completa. Esto facilita la carga rápida de datos y la actualización en tiempo real de la interfaz de usuario, mejorando significativamente la eficiencia y la fluidez de la aplicación web.

Imagen JQuery

- **GoJS**: Es una biblioteca avanzada especializada en la creación de diagramas interactivos en JavaScript. En este proyecto, GoJS desempeña un papel crucial al permitir la generación dinámica de diagramas de flujo. Estos diagramas visualizan de manera clara y comprensible los procesos empresariales y sus relaciones, lo cual es fundamental para entender y gestionar los flujos de trabajo dentro de la empresa. GoJS ofrece herramientas poderosas para personalizar y manipular los diagramas de manera interactiva, facilitando así la representación visual de datos complejos y facilitando la toma de decisiones informadas

Imagen GoJS1

2.3. Lenguajes de programación utilizados

Los códigos proporcionados utilizan varios lenguajes de programación y tecnologías para diferentes partes del desarrollo web y la interacción cliente-servidor. Aquí está una explicación breve de los principales lenguajes y tecnologías utilizadas:

2.3.1. C#

Utilizado en: Default.aspx.cs

Descripción: C# es un lenguaje de programación orientado a objetos ampliamente utilizado en el desarrollo de aplicaciones web ASP.NET. En el archivo Default.aspx.cs, C# maneja la lógica del servidor, procesa las solicitudes enviadas por los clientes y puede interactuar con bases de datos y otros sistemas externos.

Características principales:

- **Orientación a objetos:** Permite estructurar programas utilizando conceptos como clases, objetos, herencia y polimorfismo.

- **Tipado fuerte:** Requiere que todos los tipos de datos sean explícitamente declarados y no permite conversiones implícitas que puedan perder datos.

- **Manejo de excepciones:** Ofrece mecanismos para detectar y responder a errores durante la ejecución del programa, mejorando la robustez y confiabilidad del código.

- **Amplio soporte de Microsoft:** Integrado en el entorno de desarrollo de Microsoft, facilita el desarrollo de aplicaciones para el ecosistema Windows.

C# es esencial para el desarrollo de aplicaciones empresariales y de servidor, gracias a su combinación de poderosas características orientadas a objetos y su integración con herramientas y tecnologías de Microsoft.

2.3.2. JavaScript

Utilizado en: Funciones.js

Descripción: JavaScript es un lenguaje de programación interpretado ampliamente utilizado para agregar dinamismo y funcionalidad interactiva a las páginas web. En el archivo Funciones.js, JavaScript se encarga de manejar eventos del lado del cliente, realizar validaciones de formularios, manipular el DOM para actualizar dinámicamente el contenido de la página, y gestionar comunicaciones asíncronas con el servidor utilizando AJAX.

Características principales:

- **Interactividad dinámica**: Permite a las páginas web responder a la interacción del usuario en tiempo real, mejorando la experiencia de usuario.

- **Tipado dinámico**: No requiere declarar el tipo de datos de las variables antes de usarlas, lo que facilita la flexibilidad, pero requiere un manejo cuidadoso de los tipos.

- **Manipulación del DOM**: Facilita la modificación y actualización del contenido y la estructura de una página web en tiempo de ejecución.

- **Soporte para funciones de alto nivel**: Permite trabajar con funciones como objetos de primera clase, lo que favorece el desarrollo de código modular y reutilizable. JavaScript es esencial en el desarrollo web moderno por su capacidad para crear interfaces de usuario interactivas y dinámicas, así como por su integración con tecnologías clave como AJAX para la comunicación asíncrona con el servidor.

2.4. Funcionalidades Clave

El proyecto ofrece diversas funcionalidades que potencian la manipulación y visualización de datos empresariales:

- **Alternancia de Vistas Dinámicas:** Permite a los usuarios cambiar entre diferentes representaciones visuales como tablas de datos, diagramas de flujo y diagrama de tortuga, adaptándose a las necesidades específicas de visualización y análisis.

- **Interacción Avanzada con Datos:** Incluye validación de formularios antes de la carga de datos, procesamiento de archivos Excel para generar tablas dinámicas y facilitar la comprensión de conjuntos complejos de información, mejorando la precisión y la utilidad de los datos presentados.

- **Generación y Personalización de Diagramas:** Utilizando GoJS, el proyecto permite la creación de diagramas detallados y personalizables que se ajustan a las necesidades específicas de los procesos empresariales representados, proporcionando herramientas visuales poderosas para la planificación y optimización de flujos de trabajo.

- **Exportación y Descarga de Datos:** Facilita la exportación de datos en formato JSON y la descarga de imágenes de diagramas generados, permitiendo a los usuarios compartir, almacenar y utilizar la información de manera conveniente y efectiva.

2.5. Proyectos relacionados

Este proyecto se sitúa en un campo dinámico y en evolución de visualización de datos y gestión de flujos de trabajo, influenciado por proyectos innovadores que han transformado cómo las organizaciones manejan y visualizan datos complejos. Algunos ejemplos relevantes incluyen:

- **Apache Airflow**: Es una plataforma de gestión de flujos de trabajo ampliamente utilizada en entornos de datos. Permite la programación, monitoreo y coordinación de flujos de trabajo complejos mediante el uso de Python. Airflow facilita la integración con diversas herramientas de procesamiento de datos y almacenamiento, lo que lo convierte en una opción robusta para automatizar y gestionar flujos de trabajo en grandes volúmenes de datos.

Imagen Apache Airflow

- **Tableau**: Como herramienta líder en visualización de datos, Tableau permite a los usuarios crear informes interactivos y paneles de control sin necesidad de programación. (“Guía completa: Cómo utilizar Python en Power BI y sus beneficios”) Su enfoque basado en interfaz gráfica facilita la creación de visualizaciones avanzadas que ayudan a los equipos empresariales a explorar y entender sus datos de manera intuitiva. Tableau es conocido por su capacidad para manejar grandes conjuntos de datos y ofrecer opciones flexibles para análisis visual y presentación de resultados.

Imagen Tableau

- **Microsoft Power BI**: Esta suite de análisis de negocios proporciona a las organizaciones la capacidad de visualizar datos y compartir información a través de dashboards interactivos y reportes dinámicos. Power BI se destaca por su integración con otras herramientas de Microsoft, su facilidad de uso y su capacidad para conectarse a una amplia gama de fuentes de datos. (“Preguntas de entrevista de desarrollador Power BI”) Esto permite a los usuarios crear visualizaciones personalizadas y reportes automatizados que impulsan la toma de decisiones estratégicas.

Imagen Power Bi

- **Microsoft Visio**: Una herramienta especializada en la creación de diagramas que ayuda a visualizar y comunicar información compleja y procesos empresariales de manera clara y efectiva. Visio ofrece una amplia gama de plantillas y formas predefinidas que facilitan la creación de diagramas de flujo, organigramas, mapas y otros tipos de representaciones visuales. Es utilizada en entornos empresariales para documentar procesos, diseñar arquitecturas de sistemas y planificar proyectos.

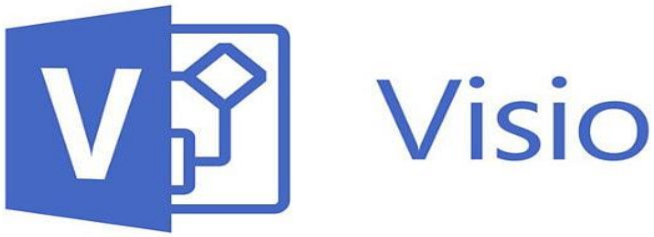


Imagen Microsoft Visio

Estos proyectos representan avances significativos en la integración de tecnologías avanzadas para la visualización dinámica de datos y la gestión eficiente de flujos de trabajo empresariales. Cada uno utiliza tecnologías y enfoques específicos para abordar necesidades particulares en cuanto a visualización, análisis y automatización de procesos, contribuyendo así a mejorar la toma de decisiones y la eficiencia operativa en diversas industrias y contextos organizacionales.

3. Fundamentos

En este apartado vamos a explicar los diferentes tipos de diagramas recreados durante este proyecto en la sección 3.1 y 3.2,

3.1. Diagrama de Flujo

Un diagrama de flujo es una representación visual que describe el flujo de control de un proceso, sistema o algoritmo. Se utiliza ampliamente en diversas disciplinas como la ingeniería de software, la gestión de proyectos, la ingeniería industrial, entre otras. Consiste en símbolos gráficos que representan diferentes tipos de pasos o actividades dentro del proceso, junto con flechas que indican la dirección del flujo y conexiones entre estos pasos. Los diagramas de flujo son útiles para visualizar de manera clara y comprensible cómo se ejecuta un proceso, desde el inicio hasta la conclusión, mostrando decisiones, operaciones, ramificaciones y bucles.

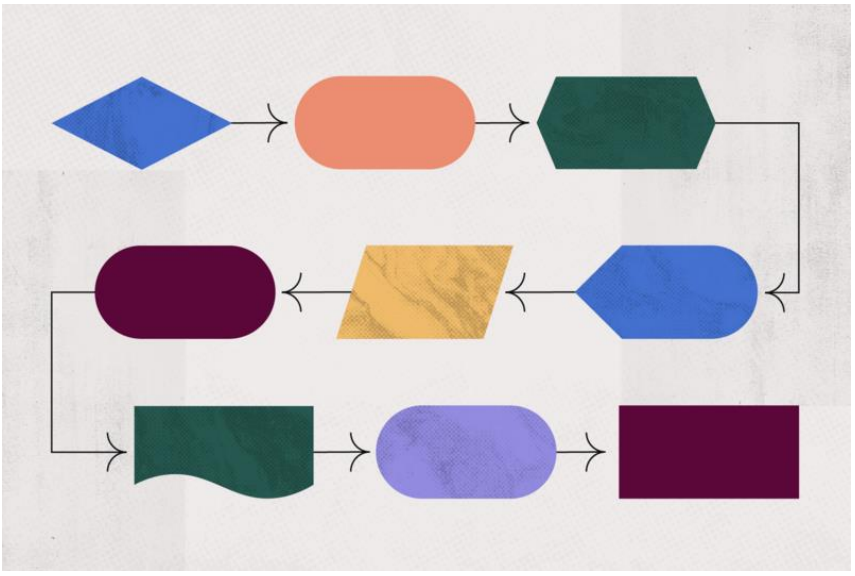


Imagen Diagrama Flujo

3.2. Diagrama de Tortuga

(Safety Culture, 2024)"Los diagramas de tortuga se utilizan habitualmente en proyectos de mejora de procesos empresariales para trazar el estado actual de un proceso e identificar oportunidades de mejora."
(Safety Culture, 2024) También pueden utilizarse para trazar el estado futuro de un proceso una vez realizadas las mejoras. "Son una herramienta valiosa para cualquier organización que quiera crecer y mejorar sus procesos empresariales."

2Imagen Diagrama Tortuga

3.3. Microsoft Visual Studio

Visual Studio es un entorno integrado de desarrollo (IDE) desarrollado por Microsoft, ampliamente reconocido por su versatilidad en la creación de una variedad de aplicaciones, ya sean de escritorio, web, móviles o de otro tipo. Este software ofrece una amplia gama de herramientas y servicios diseñados para facilitar todas las etapas del ciclo de vida del desarrollo de software. Desde la fase inicial de escribir y depurar el código hasta la etapa final de compilación, pruebas, despliegue y mantenimiento de las aplicaciones, Visual Studio se destaca por su robustez y capacidad para mejorar la productividad de los desarrolladores.

Imagen Microsoft Visual Studio

Una de las características distintivas de Visual Studio es su capacidad de integración con numerosos lenguajes de programación y plataformas, lo que permite a los desarrolladores trabajar en proyectos diversificados con eficiencia y coherencia. Además, ofrece herramientas avanzadas de depuración que permiten identificar y corregir errores de manera efectiva, mejorando así la calidad del software desarrollado. Con soporte para múltiples frameworks y bibliotecas, los desarrolladores pueden aprovechar al máximo las capacidades de Visual Studio para crear aplicaciones que cumplan con los estándares más exigentes del mercado.

Además de su potencia como IDE, Visual Studio se destaca por su comunidad activa y el soporte continuo de Microsoft, que garantiza actualizaciones regulares y nuevas características que mantienen a los desarrolladores al tanto de las últimas tendencias y tecnologías emergentes. Esto no solo promueve la innovación en el desarrollo de software, sino que también ayuda a los equipos a mantenerse competitivos en un entorno dinámico y en constante evolución.

3.4. JSON (JavaScript Object Notation)

JSON, acrónimo de JavaScript Object Notation, se ha establecido como un estándar fundamental en el intercambio de datos debido a su simplicidad y versatilidad. Este formato ligero y legible para humanos facilita tanto la escritura y lectura de datos como su interpretación por máquinas. Utilizado principalmente en el desarrollo de aplicaciones web y servicios web, JSON permite la transferencia eficiente de información entre sistemas y plataformas diversas y heterogéneas.

En la estructura de un objeto JSON, los datos se organizan en pares clave-valor, donde las claves son cadenas de texto y los valores pueden ser de cualquier tipo válido en JSON: números, cadenas, booleanos, arrays o incluso otros objetos JSON. Esta flexibilidad lo hace ideal para representar datos estructurados de manera clara y concisa. Es particularmente adecuado para la comunicación entre el frontend y el backend de aplicaciones web, facilitando una integración fluida y eficiente.

Además de su uso en la comunicación entre sistemas, JSON también encuentra aplicación en el almacenamiento de datos en bases de datos NoSQL y otros sistemas de almacenamiento que soportan datos semiestructurados o no estructurados. Esto permite a los desarrolladores aprovechar su capacidad para manejar información compleja de manera eficiente y escalable, adaptándose a las necesidades cambiantes del desarrollo de software moderno.

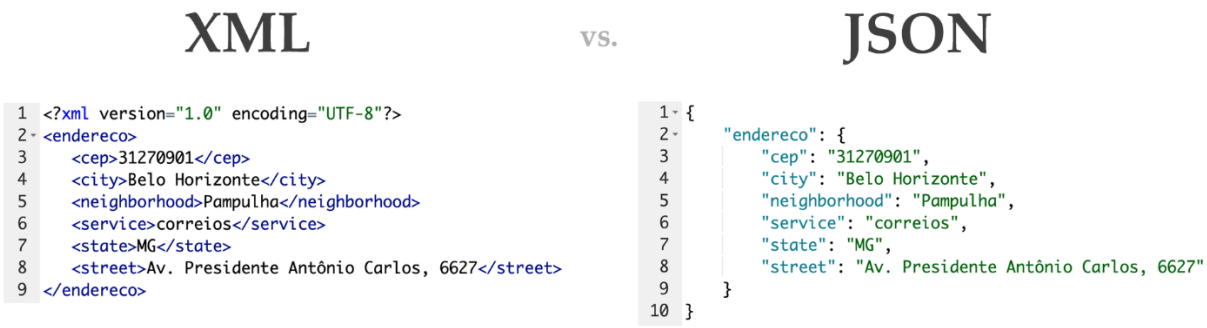


Imagen JSON

4. Metodología

Durante el desarrollo de este proyecto, se siguieron pasos estructurados y metódicos para asegurar un progreso efectivo y eficiente, centrado en los objetivos y requisitos específicos del mismo. A continuación, se detalla la metodología aplicada:

4.1. Aprendizaje de C# para la Lectura de un Excel

Objetivo: Adquisición de Competencias en C# para Manipular y Leer Datos desde Archivos Excel y Convertir a JSON

Para alcanzar competencias sólidas en C# relacionadas con la manipulación de datos desde archivos Excel y su conversión a JSON, se llevó a cabo un proceso estructurado que comprendió investigación, implementación y pruebas exhaustivas.

- **Investigación de Métodos de Lectura**

  Se inició con una exhaustiva exploración de las diversas técnicas y métodos disponibles en C# para la lectura eficiente de datos desde archivos Excel. El objetivo fue identificar y seleccionar el enfoque más adecuado según los requerimientos del proyecto. Esta fase implicó estudiar bibliotecas y métodos nativos de C# para manipular archivos Excel, evaluando su eficiencia y capacidad de manejar grandes volúmenes de datos de manera precisa.

- **Implementación y Pruebas**

  Una vez definido el método de lectura más apropiado, se procedió a desarrollar un código en C# que fuera capaz de extraer datos específicos desde los archivos Excel utilizados. Durante la implementación, se puso un énfasis significativo en la estructura del código para garantizar la legibilidad, modularidad y eficiencia del proceso de extracción de datos. Posteriormente, se realizaron pruebas exhaustivas para verificar la precisión y fiabilidad de la extracción, asegurando que los datos se manipularan correctamente según los requisitos del proyecto.

- **Uso de Newtonsoft para Transformar los Datos a Formato JSON**

  Como parte integral del proceso de transformación de datos, se integró la reconocida librería Newtonsoft.JSON en el entorno de desarrollo en C#. Esta librería permitió convertir los datos obtenidos desde los archivos Excel al formato JSON de manera eficiente y con alta precisión. Se implementaron funciones específicas para serializar y deserializar datos entre Excel y JSON, facilitando así la interoperabilidad y el intercambio de datos entre diferentes sistemas y plataformas.

4.2. Conexión entre FrontEnd y BackEnd utilizando AJAX (Asynchronous JavaScript And XML)

Objetivo: Establecer una comunicación eficiente entre la interfaz de usuario (FrontEnd) y el servidor (BackEnd) utilizando tecnologías web como JavaScript y AJAX para mejorar la interactividad y la experiencia del usuario.

- **Desarrollo de Endpoints en el BackEnd**

  Se comenzó implementando endpoints en el servidor utilizando C#, diseñados para manejar de manera segura y eficiente las solicitudes HTTP provenientes del FrontEnd. Estos endpoints actúan como interfaces que permiten al FrontEnd interactuar con la lógica de negocio y los datos almacenados en el servidor, siguiendo principios de seguridad y buenas prácticas de desarrollo web.

- **Uso de AJAX en el FrontEnd**

  Para facilitar la comunicación asíncrona entre el FrontEnd y el BackEnd, se empleó JavaScript en conjunto con AJAX. Esta combinación tecnológica posibilita realizar solicitudes al servidor sin necesidad de recargar la página completa, lo que mejora la respuesta y la fluidez de la aplicación web. El uso de AJAX permitió actualizar dinámicamente el contenido en la interfaz de usuario en respuesta a las acciones del usuario o eventos ocurridos en el servidor, proporcionando una experiencia interactiva y fluida.

- **Integración y Pruebas de Comunicación**

  Una vez implementada la comunicación entre el FrontEnd y el BackEnd mediante AJAX, se realizaron pruebas exhaustivas para validar su funcionamiento correcto. Durante las pruebas, se verificó la transmisión adecuada de datos entre las capas FrontEnd y BackEnd, asegurando que la información se recibiera y procesara correctamente en ambos extremos. Además, se evaluó la gestión de errores y excepciones para garantizar la robustez y fiabilidad del sistema en diversas condiciones de uso.

4.3. Desarrollo de la interfaz de usuario con HTML y Semantic UI

Objetivo: Diseñar una interfaz de usuario moderna, intuitiva y estéticamente agradable utilizando HTML y Semantic UI, centrada en mejorar la experiencia del usuario en la aplicación web.

- **Uso de HTML**

  Se utilizó HTML como lenguaje base para estructurar y organizar los elementos de la interfaz de usuario de manera semántica y eficiente. HTML proporcionó la estructura fundamental para la disposición jerárquica de los componentes visuales, facilitando la creación de una interfaz clara y bien organizada que permite a los usuarios navegar e interactuar de manera intuitiva.

- **Implementación de Semantic UI**

  Para asegurar una apariencia coherente y profesional, se implementaron los principios de diseño de Semantic UI. Esta biblioteca CSS ofrece una amplia gama de componentes predefinidos y estilos visuales que se alinean con el esquema de colores de EGAM. Al emplear Semantic UI, se diseñaron botones, formularios, barras de navegación y otros elementos visuales de manera consistente, mejorando la usabilidad y la estética general de la interfaz de usuario.

- **Validación y Optimización**

  Se llevaron a cabo pruebas exhaustivas de usabilidad y accesibilidad para validar el diseño de la interfaz de usuario. Durante este proceso, se evaluó cómo los usuarios interactúan con la interfaz y se identificaron áreas de mejora en términos de navegación, legibilidad y fluidez de la experiencia de usuario. Además, se optimizó el diseño y la funcionalidad de acuerdo con las mejores prácticas de diseño web actuales, asegurando que la interfaz no solo fuera visualmente atractiva, sino también funcional y accesible para todos los usuarios.

4.4. Evaluación y selección de bibliotecas de JavaScript para la visualización de diagramas

Objetivo: Investigar y seleccionar la mejor biblioteca de JavaScript para la visualización dinámica de diagramas dentro del proyecto, asegurando una representación gráfica efectiva y eficiente de los datos.

- **Análisis de Bibliotecas Disponibles**

  Se realizó una investigación exhaustiva de varias bibliotecas populares, como Mermaid.JS, JointJS, D3.JS y GoJS. Cada una fue evaluada en función de sus capacidades específicas para la representación gráfica de datos, teniendo en cuenta aspectos como la flexibilidad, la capacidad para manejar diagramas complejos, la integración con otras tecnologías y la documentación disponible. Este análisis permitió obtener una visión clara de las fortalezas y limitaciones de cada biblioteca en relación con los requisitos del proyecto.

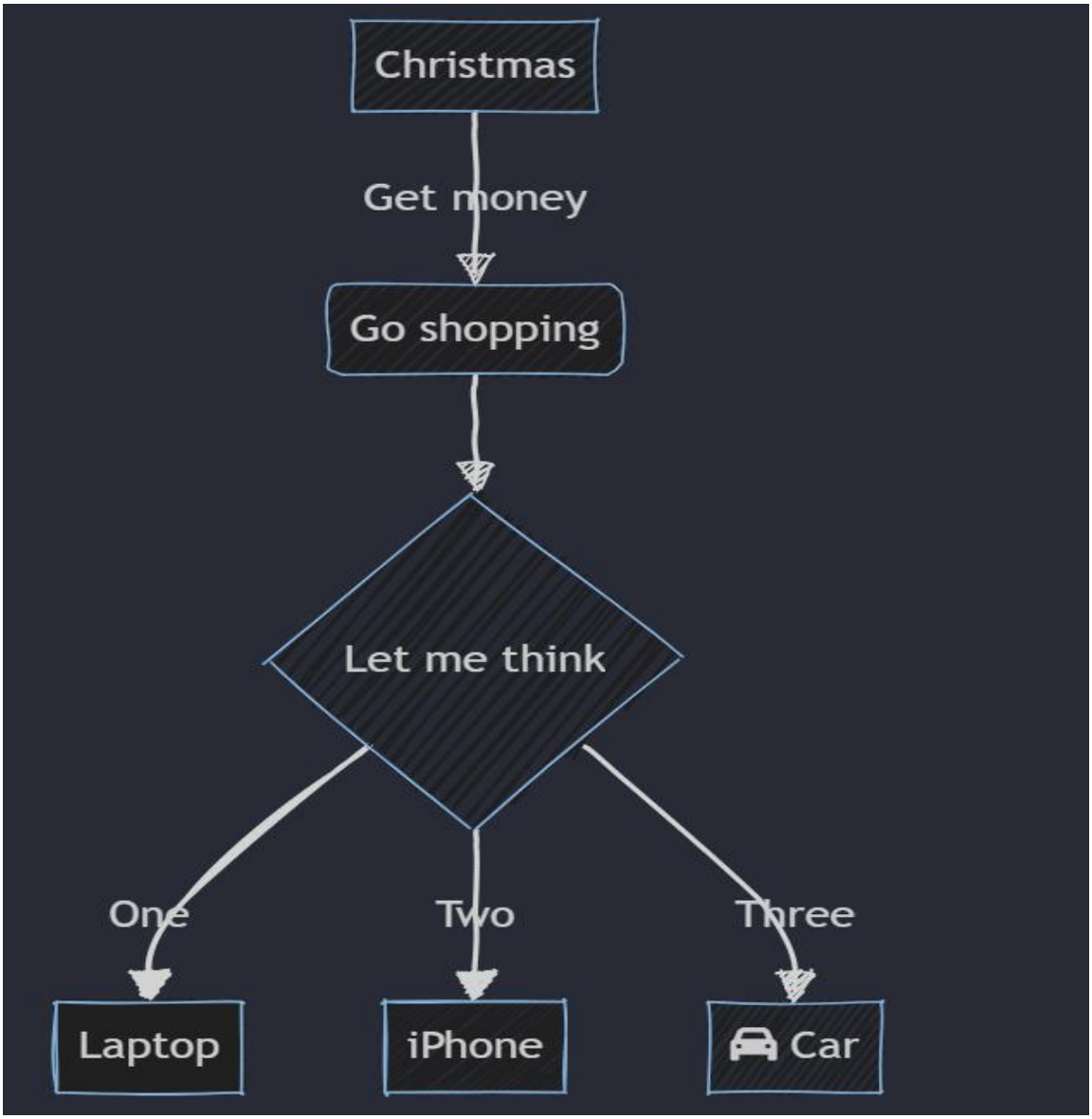


Imagen ejemplo mermaid.js

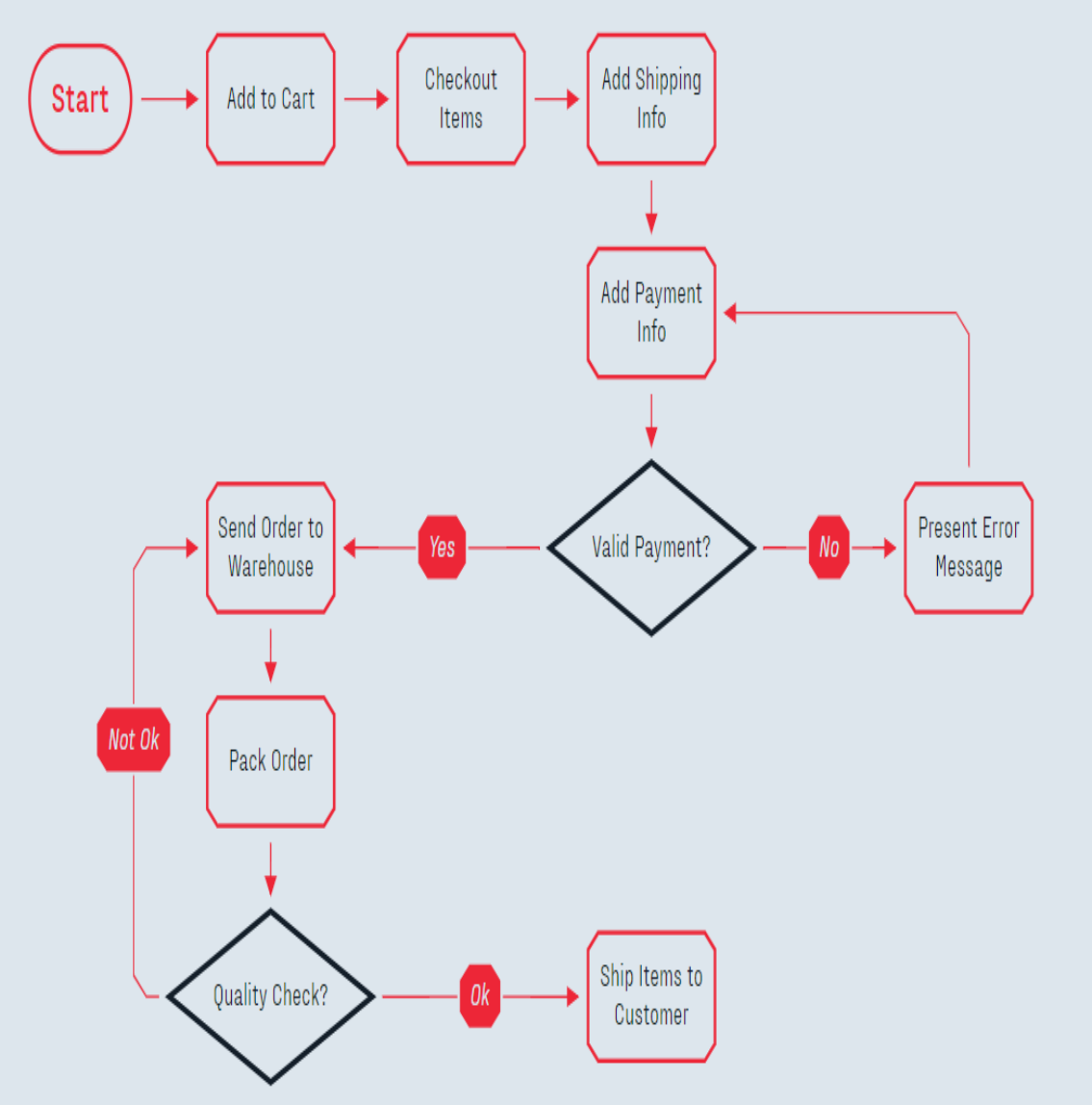


Imagen Ejemplo JointJs

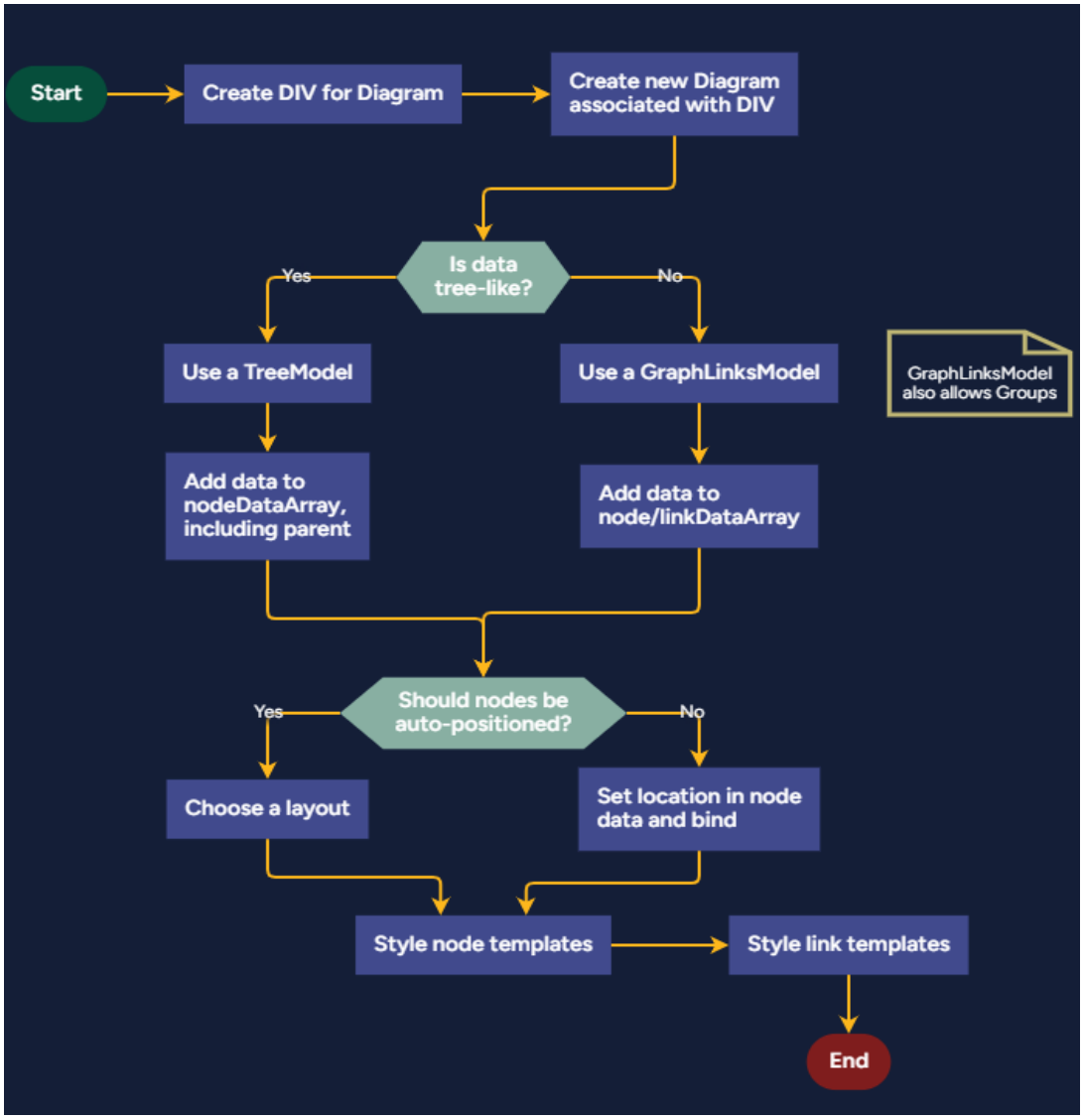


Imagen ejemplo GoJS

- **Prototipado y Evaluación**

  Se procedió a la implementación de prototipos utilizando las bibliotecas seleccionadas. Durante esta fase, se evaluó el rendimiento de cada biblioteca en términos de velocidad de renderizado, capacidad de personalización y facilidad de integración con el entorno de desarrollo existente. Los prototipos también fueron utilizados para simular escenarios prácticos y verificar la capacidad de cada biblioteca para cumplir con los requisitos específicos del proyecto, incluyendo la visualización de diagramas complejos y la interacción dinámica con los datos.

- **Selección de GoJS**

Tras comparar y analizar detalladamente las diferentes opciones, se seleccionó GoJS como la biblioteca más adecuada para el proyecto. GoJS destacó por su robustez y eficiencia en el manejo de diagramas complejos, así como por su amplia gama de características que facilitan la creación y manipulación de gráficos interactivos. Además, la integración fluida de GoJS con el entorno de desarrollo existente y su soporte activo fueron factores determinantes en la decisión final. Esta elección aseguró que el proyecto contara con una herramienta potente y versátil para la visualización dinámica de datos mediante diagramas, cumpliendo con los estándares de rendimiento y calidad esperados.

5. Descripción del conjunto del código

El conjunto de código se divide en tres archivos primarios: Default.aspx, Default.aspx.cs, y Funciones.js. El código se enfoca en la gestión y visualización de datos en una aplicación web.

Default.aspx define la estructura y los elementos HTML de la página, incluyendo formularios y contenedores para mostrar diagramas y tablas. Default.aspx.cs es el código detrás del archivo ASP.NET que maneja la lógica del servidor, como la carga y procesamiento de archivos Excel, y la generación de datos en formato JSON para ser enviados al cliente. Funciones.js maneja la lógica de la interfaz de usuario en el lado del cliente, incluyendo la inicialización de formularios y botones, el manejo de eventos, las solicitudes AJAX al servidor, y la generación de diagramas de flujo y tortuga usando la biblioteca GoJS.

5.1. Default.aspx

Código que define una página web en ASP.NET que permite a los usuarios cargar un archivo Excel, visualizar datos, y descargarlos en formato JSON. La interfaz incluye un formulario con botones para cargar el archivo, descargar JSON, y descargar un diagrama como imagen. Hay contenedores para mostrar la tabla de datos (excelContainer) y dos contenedores adicionales (grafoContainer y tortugaContainer) que se pueden alternar mediante botones. Se utilizan varias bibliotecas JavaScript y CSS, como jQuery, GoJS, Semantic UI y html2canvas, para la manipulación y visualización de datos y gráficos.

5.2. Default.aspx.cs

Este código en C# define la lógica del lado del servidor para una página web ASP.NET que permite la carga de archivos Excel, la extracción y procesamiento de sus datos, y la generación de una respuesta JSON.

Cuando un archivo Excel es subido, es guardado en el servidor y leído utilizando la biblioteca NPOI. Los datos se extraen de varias hojas del Excel para formar tablas, diagramas de procesos (gráficos), entradas, salidas, procesos, indicadores, usuarios y procedimientos, y se convierten en un formato JSON utilizando "Newtonsoft.Json." Esta información se envía de vuelta al cliente en la respuesta HTTP.

5.3. Funciones.js

Este código implementa una interfaz de usuario interactiva para gestionar y visualizar datos mediante diagramas de flujo y diagramas de tortuga, utilizando bibliotecas como jQuery y GoJS. La interfaz permite al usuario subir archivos, descargar datos en formato JSON o imagen, y cambiar entre diferentes vistas (Excel, grafo y tortuga). Al cargar datos, se generan visualizaciones dinámicas: un diagrama de flujo para representar nodos y enlaces, y un diagrama de tortuga para mostrar un proceso y sus componentes asociados. La función "handleOnSubmit" gestiona el envío de formularios, mientras que "MostrarDatos" y otras funciones manejan la visualización de los datos cargados.

5.4. Explicación de Cada Método, Clase o Componente

5.4.1. Default.aspx

5.4.1.1. Directiva

- **Línea 1:** Esta línea es una directiva de ASP.NET que define el lenguaje de la página (C#), habilita el auto enlace de eventos (AutoEventWireup="true"), especifica el archivo de código subyacente (CodeBehind="Default.aspx.cs") y la clase que hereda (Inherits="ABDFinal.WebForm1").

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ABDFinal.WebForm1" %>
```

5.4.1.2. Inicio HTML

- **Línea 3**
  Se define el inicio del documento HTML.

- **Línea 3**
  La etiqueta <html> especifica el uso de XHTML.
- **Línea 5**
  La etiqueta <head> incluye varios elementos importantes.

- **Línea 6**
  "meta" para especificar el tipo de contenido y la codificación de caracteres.

- **Líneas 7-11**
  Se cargan varios scripts de JavaScript y CSS, incluyendo jQuery, GoJS, Semantic UI, DoubleTreeLayout y html2canvas.

- **Línea 12**
  Se incluye un script personalizado (Funciones.js) con un parámetro de consulta basado en la fecha y hora actual para evitar el almacenamiento en caché.

- **Línea 13**
  Un enlace a una hoja de estilos (semantic.min.css).

- **Línea 14**
  Se define el título de la página como "Data Table".

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="Script/jquery-3.7.1.min.js"></script>
    <script src="https://unpkg.com/gojs/release/go-debug.js"></script>
    <script src="Script/semantic.min.js"></script>
    <script src="Script/DoubleTreeLayout.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js
"></script>
    <script
src="Script/Funciones.js?t=<%=DateTime.Now.ToString("yyyyMMddHHmmss_ff
fffff") %>"></script>
    <link href="CSS/semantic.min.css" rel="stylesheet" />
    <title>Data Table</title>
```

5.4.1.3. Estilo CSS
5.4.1.4. Líneas 15-136

- Estilos básicos para el cuerpo (body).

- Estilos para el contenido (#content), formularios (form.ui.form), botones (.ui.red.primary.submit.button), contenedores (#excelContainer, .ui.segment), tablas (table.ui.striped.table th, table.ui.striped.table td), y otros elementos.

- Específicos de la clase (cls-1, cls-2, cls-3, cls-4) utilizados en el SVG del logotipo.

- Estilos para el logotipo (.logo-egam, .power-by, .capa-svg).
- Estilos para botones deshabilitados y habilitados (button[disabled], .ui.button[disabled], .ui.red.primary.submit.button[disabled], .ui.button, .ui.button:hover, .ui.button.enabled).

```
<style>
    body {
        background-color: #f5f5f5;
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        margin: 0;
        padding: 0;
        display: flex;
        min-height: 100vh;
    }

    #content {
        flex: 1;
        padding: 20px;
    }

    form.ui.form {
        background-color: #fff;
        padding: 20px;
        border-radius: 5px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        max-width: 100%;
        margin: auto;
    }

    .ui.red.primary.submit.button {
        background-color: #db2828;
        color: #fff;
        margin-top: 10px;
    }

    #excelContainer {
        margin-top: 20px;
    }

    .ui.segment {
        position: relative;
    }

        .ui.segment h2.ui.header {
            text-align: center;
        }
```

```css
table.ui.striped.table th,
        table.ui.striped.table td {
            border: 1px solid #ddd;
            padding: 10px;
            text-align: left;
        }
        table.ui.striped.table th {
            background-color: #8080809c;
        }
        table.ui.striped.table tbody tr:nth-child(even) {
            background-color: #f9f9f9;
        }

        table.ui.striped.table tbody tr:hover {
            background-color: #f1f1f1;
        }

        .cls-1, .cls-2, .cls-4 {
            fill-rule: evenodd;
        }

        .cls-2 {
            fill: #DC0A2D;
        }

        .cls-3 {
            fill: #202020;
            stroke: #000;
            stroke-width: 1px;
        }

        .cls-4 {
            fill: #0D0D0D;
        }

        .logo-egam {
            position: absolute;
            top: 10px;
            right: 10px;
            display: flex;
            align-items: center;
        }

        .power-by {
            top: 5px;
            font-size: 0.8em;
            color: #666;
            margin-right: 10px;
        }

        .capa-svg {
            font-size: 0.8em;
            color: #666;
        }
```

```
    button[disabled], .ui.button[disabled], .ui.red.primary.submit.button[disabled] {
        background-color: #d3d3d3 !important;
        color: #808080 !important;
    }


      .ui.red.primary.submit.button  {
        background-color: #db2828 !important;
        color: #fff !important;
    }

    .ui.button {
        background-color: #db2828;
        color: white;
    }

    .ui.button:hover {
        background-color: #ff6961;
    }

    .ui.button.enabled {
        background-color: #ff6961;
    }
</style>
```

5.4.1.5. Body

- **Línea 139**
  Dentro del cuerpo (<body>), se define un contenedor (#content) que ocupa toda la pantalla.

- **Línea 140**
  Se incluye un segmento ( )

- **Línea 141**
  Un encabezado (<h2 class="ui header">Tabla de Datos</h2>).

- **Líneas 142-146**
  Botones para cambiar la vista ( ), todos inicialmente deshabilitados (disabled).

- **Líneas 147-157**
  Un logotipo en la esquina superior derecha (.logo-egam) con texto "Powered by" y un SVG como logotipo.

- **Líneas 159**

Un formulario (<form class="ui form" enctype="multipart/form-data">) permite la carga de archivos y contiene:

- **Línea 160**
  Un campo de entrada para seleccionar archivos

- **Línea 161**
  Un botón para subir el archivo

- **Línea 162**
  Un botón para descargar un archivo JSON, inicialmente oculto.

- **Línea 163**
  Un botón para descargar un diagrama como imagen, inicialmente oculto.

- **Línea 164**
  Contenedor para mostrar el contenido del archivo Excel.

- **Línea 165**
  Contenedor para mostrar un grafo.

- **Línea 166**
  Contenedor para mostrar un grafo

```
<body>
    <div id="content">
        <div class="ui segment">
            <h2 class="ui header">Tabla de Datos</h2>
            <div class="ui buttons">
                <button id="btnExcel" class="ui button" disabled>Excel</button>
                <button id="btnGrafo" class="ui button" disabled>Grafo</button>
                <button id="btnTortuga" class="ui button"
disabled>Tortuga</button>
            </div>
            <div class="logo-egam">
                <div class="power-by">Powered by</div>
                <div class="capa-svg">
                    <svg data-name="Grupo 1"
xmlns="http://www.w3.org/2000/svg" width="100" height="100" viewBox="0
0 703.377 256.281">
                        <path id="e" class="cls-1"
d="M751.833,559.46q8.057,0,11.678-2.97t3.621-
9.644a38.286,38.286,0,0,0-3.214-15.055,42.944,42.944,0,0,0-8.749-
13.265,45.851,45.851,0,0,0-15.625-10.62,49.661,49.661,0,0,0-18.88-
3.54q-23.52,0-37.191,12.9t-
13.672,35.2q0,21.485,13.387,34.1t36.174,12.614q18.962,0,30.965-
6.307t12-15.829a9.238,9.238,0,0,0-3.133-7.2,11.564,11.564,0,0,0-8.016-
2.807q-4.8,0-11.8,4.476-2.118,1.385-3.256,2.034a33.98,33.98,0,0,1-
6.714,2.808,23.607,23.607,0,0,1-6.388.854q-8.871,0-13.265-4.354t-4.72-
13.3Zm-42.44-29.744q4.027-4.353,10.946-
4.354,7.4,0,11.433,4.15t4.6,12.289H705.039Q705.363,534.07,709.393,529
.716Z" transform="translate(-669.813 -411.563)"></path>
```

```
path id="GAM" class="cls-2" d="M958.988,531.3q0-17.579-7.568-25.7t-
24.17-8.118h-23.8q-13.429,0-19.836,4.944t-
6.409,15.32q0,9.4,5.31,13.794t16.785,4.394h6.714a24.607,24.607,0,0,1-
10.926,15.931q-8.851,5.922-21.545,5.92-17.7,0-26.367-12.085t-8.667-
36.5q0-26.367,8.606-39.673t25.94-
13.306q11.961,0,23.315,10.254,1.708,1.465,2.686,2.32a10.518,10.518,0,0,
0,.854.732q9.888,8.668,19.9,8.667,11.472,0,18.188-6.286t6.714-16.907q0-
18.677-19.775-31.067t-50.9-12.39q-43.212,0-68.481,25.512t-
25.269,69.214q0,44.069,22.217,69.4t60.791,25.33q18.431,0,32.166-
5.066a61.211,61.211,0,0,0,23.5-
15.32q0.243,9.156,4.883,13.672t14.16,4.517q10.254,0,15.625-
5.738t5.371-
16.6V531.3ZM1032.96,572.2h57.5l4.03,10.5a23.935,23.935,0,0,0,9.15,12.
207q6.225,4.149,15.38,4.151,11.715,0,18.86-6.714t7.14-
17.822a38.5,38.5,0,0,0-.48-6.226,23.55,23.55,0,0,0-1.35-5L1099,440.238q-
5.61-15.867-11.84-21.179t-17.21-5.31h-16.72q-10.995,0-17.52,5.737-
6.54,5.739-11.78,20.752l-43.7,123.047a19.427,19.427,0,0,0-
1.1,4.639,49.508,49.508,0,0,0-
.366,6.347q0,11.354,7.019,18.067t18.977,6.714a26.174,26.174,0,0,0,15.02
-4.212,24.29,24.29,0,0,0,9.15-12.146Zm29.06-99.975,17.94,62.256h-
36.25Zm160.64,36.987q0-.243.12-5.005t0.12-9.155q0-6.836-.12-10.986t-
0.36-
8.057q2.8,14.282,4.33,21.484t2.99,12.574l18.92,72.631q2.205,8.67,7.45,1
2.207t15.87,3.54q10.62,0,16.17-3.723t7.75-12.268l18.8-74.463q1.47-
5.37,3.3-13.305t4.15-18.677q-0.12,10.011-.43,20.447t-
0.3,11.047V572.44q0,12.574,7.01,19.288t19.96,6.713q12.21,0,18.5-
6.774t6.28-19.959V439.018q0-12.205-6.77-18.616t-19.72-6.409h-21.24q-
11.835,0-18.06,5.188t-9.89,18.372l-19.53,68.725q-1.23,4.029-2.5,9.7-
1.29,5.676-3.24,16.3-2.445-11.106-3.97-17.7t-2.13-8.667l-18.92-67.139q-
4.275-15.134-10.2-20.2t-18.37-5.066h-21.24q-12.945,0-19.71,6.409-
6.78,6.408-
6.78,18.615V571.708q0,13.183,6.29,19.959t18.49,6.774q12.945,0,19.96-
6.713t7.02-19.288V509.208Z" transform="translate(-669.813  -
411.563)"></path>
                    <circle id="Elipse_1" data-name="Elipse 1" class="cls-3"
cx="40.187" cy="203.437" r="8"></circle>
                  </svg>
               </div>
            </div>
         </div>


         <form class="ui form" enctype="multipart/form-data"  style="left: 8px;
top: -1px">
            <input type="file" name="facturaPeritacion" id="file"
accept="application/vnd.ms-excel,application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"  />
            <div class="ui red primary submit button" id="btnUploadFile">Subir
Archivo</div>
```

```
            <div class="ui red primary submit button" id="btnDownloadJSON"
style="display: none">Descargar JSON</div>
            <div id="btnDownloadImage" class="ui red primary submit button"
style="display: none;">Descargar diagrama como imagen</div>
            <div id="excelContainer" runat="server" class="ui grid"
style="margin-top: 20px; height: 686px;"></div>
            <div id="grafoContainer" style="width: 100%; height: 800px; display:
none;"></div>
            <div id="tortugaContainer" style="width: 100%; height: 800px;
display: none;"></div>
        </form>
    </div>
</body>
</html>
```

5.4.2. Funciones.js

5.4.2.1. initializeForm: Esta función inicializa el formulario utilizando la biblioteca de UI jQuery. Asocia el evento de envío del formulario a la función "handleOnSubmit".

- **Línea 15**
  Inicializa el formulario usando la configuración predeterminada de Semantic UI.

- **Línea 16**
  Adjunta el evento submit del formulario a la función "handleOnSubmit" para manejar el envío del formulario.

```
function initializeForm() {
    $('.ui.form').form();
    $('.ui.form').on('submit',  handleOnSubmit);
}
```

5.4.2.2. initializeButtons: Configura los eventos de clic para los botones de la interfaz de usuario.

- **Línea 20**:
  Muestra el contenedor de Excel cuando se hace clic en el botón de Excel.

- **Línea 23**
  Muestra el contenedor del grafo cuando se hace clic en el botón de grafo.

- **Línea 26**
  Muestra el contenedor de tortuga cuando se hace clic en el botón de tortuga.

- **Línea 29**
  Llama a la función “uploadFile” cuando se hace clic en el botón de subir archivo.

- **Línea 32**
  Llama a la función “downloadJSON” cuando se hace clic en el botón de descargar JSON.

- **Línea 35**
  Llama a la función “downloadImage” cuando se hace clic en el botón de descargar imagen.

```
function initializeButtons() {
    $('#btnExcel').on('click', function () {
        showContainer('excel');
    });
    $('#btnGrafo').on('click', function () {
        showContainer('grafo');
    });
    $('#btnTortuga').on('click', function () {
        showContainer('tortuga');
    });
    $('#btnUploadFile').on('click', function () {
        uploadFile();
    });
    $('#btnDownloadJSON').on('click', function () {
        downloadJSON();
    });
    $('#btnDownloadImage').on('click', function () {
        downloadImage();
    });
}
```

5.4.2.3. showContainer: Muestra el contenedor especificado (excel, grafo, tortuga) y oculta los demás.

- **Línea 41**

Oculta todos los contenedores.

- **Líneas 45-46**

Muestra el contenedor de Excel si el tipo es excel.

- **Líneas 46-53** Muestra el contenedor de grafo y genera el diagrama de flujo si hay datos de grafo válidos.

- **Líneas 54-61** Muestra el contenedor de tortuga y genera el diagrama de tortuga si hay datos disponibles.

```
function showContainer(type) {
    $('#excelContainer, #grafoContainer, #tortugaContainer').hide();

    console.log("Switching to view:", type);

    if (type === 'excel') {
        $('#excelContainer').show();
    } else if (type === 'grafo') {
        $('#grafoContainer').show();
        if (datosObtenidos && datosObtenidos.Grafo) {
            generarDiagramaFlujo(datosObtenidos.Grafo);
        } else {
            console.error('Invalid or missing graph data.');
        }
    } else if (type === 'tortuga') {
        $('#tortugaContainer').show();
        if (datosObtenidos) {
            generarDiagramaTortuga(datosObtenidos);
        } else {
            generarDiagramaTortuga();
        }
    }
}
```

5.4.2.4. **MostrarDatos**: Muestra inicialmente la tabla de datos.

- **Linea65**

Muestra los datos en una tabla

```
function MostrarDatos(data, vistaTabla) {
        showTable(data.datosExcell);
}
```

5.4.2.5. showTable: Construye y muestra una tabla HTML con los datos proporcionados.

- **Línea 69**

Inicia la construcción de la tabla HTML.

- **Líneas 71-74**

Crea los encabezados de la tabla.

- **Líneas 76-82**

Crea las filas de la tabla con los datos.

- **Línea 84**

Finaliza la construcción de la tabla HTML.

- **Líneas 86-87**

la tabla en el contenedor de Excel.

```
function showTable(datosExcell) {
    let html = "<table class='ui striped table'><thead><tr>";
    for (let key in datosExcell) {
        html += `<th>${key}</th>`;
    }
    html += "</tr></thead><tbody>";
    for (let i = 0; i < datosExcell[Object.keys(datosExcell)[0]].length; i++) {
        html += "<tr>";
        for (let key in datosExcell) {
            html += `<td>${datosExcell[key][i]}</td>`;
        }
        html += "</tr>";
    }
    html += "</tbody></table>";
    $("#excelContainer").html(html);
    showContainer('excel');
}
```

5.4.2.6. handleOnSubmit: Maneja el envío del formulario y envía los datos al servidor.

- **Línea 91**

Previene el comportamiento predeterminado del formulario.

- **Línea 92**

Verifica si el formulario es válido.

- **Línea 93-95**

Crea un objeto FormData con los datos del formulario y añade un parámetro adicional.

- **Linea 97-105**

Envía una solicitud AJAX al servidor.

```
function handleOnSubmit(e) {
    e.preventDefault();
    if ($('.ui.form').form('is  valid')) {
        let form = document.querySelector('.form');
        let Datos = new FormData(form);
        Datos.append("op", "1");

        $.ajax({
            type: "POST",
            url: "Default.aspx",
            processData: false,
            contentType: false,
            data: Datos,
            success: handleSuccess,
            error: handleError
        });
    } else {
        console.error('Form is incomplete');
    }
}
```

5.4.2.7. handleSuccess: Procesa la respuesta exitosa del servidor y muestra los datos.

- **Línea 112 y Línea 133**

Verifica el estado de la respuesta del servidor.

- **Líneas 113-114**

Guarda los datos obtenidos y llama a MostrarDatos

.

- **Líneas 117-123**

Si el contenedor de grafo está visible, genera el diagrama de flujo.

- **Líneas 126-132**

Si el contenedor de tortuga está visible, genera el diagrama de tortuga.

```
function handleSuccess(data, txtStatus, jqXHR) {
    if (data.Estado === "ok") {
        datosObtenidos = data;
        MostrarDatos(data, true);

        if ($('#grafoContainer').is(':visible')) {
            if (datosObtenidos && datosObtenidos.Grafo) {
                generarDiagramaFlujo(datosObtenidos.Grafo);
            } else {
                console.error('Invalid or missing graph data.');
            }
        }

        if ($('#tortugaContainer').is(':visible')) {
            if (datosObtenidos) {
                generarDiagramaTortuga(datosObtenidos);
            } else {
                generarDiagramaTortuga();
            }
        }
    } else if (data.Estado === "nok") {
        console.log(data.error);
    }
}
```

5.4.2.8. handleError: Maneja errores en la solicitud AJAX.

- **Línea 139**

Muestra el error en la consola

```
function handleError(jqXHR, textStatus, errorThrown) {
    console.log(errorThrown);
}
```

5.4.2.9. downloadJSON: Permite descargar los datos en formato JSON.

- **Línea 143 y Líneas 151-153**

Verifica si hay datos disponibles para descargar.

- **Línea 144**

Crea un blob con los datos en formato JSON.

- **Líneas 146-150**

Crea un enlace de descarga y simula un clic para iniciar la descarga.

```
function downloadJSON() {
    if (datosObtenidos) {
        var blob = new Blob([JSON.stringify(datosObtenidos)], { type:
'application/json' });
        var link = document.createElement('a');
        link.href = URL.createObjectURL(blob);
        link.download = 'datos.json';
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    } else {
        console.error('No hay datos disponibles para descargar');
    }
}
```

5.4.2.10. downloadImage: Permite descargar una imagen del diagrama visible.

- **Línea 157**

Determina qué contenedor está visible.

- **Líneas 159-162**

Verifica si hay un diagrama visible.

- **Línea 164**

Utiliza html2canvas para convertir el contenedor en un lienzo.

- **Líneas 165-172**

Crea un enlace de descarga y simula un clic para iniciar la descarga de la imagen.

```
function downloadImage() {
    let container = $('#grafoContainer').is(':visible') ? $('#grafoContainer') :
$('#tortugaContainer');

    if (!container.is(':visible')) {
        console.error('No hay diagrama visible para descargar');
        return;
    }

    html2canvas(container[0]).then(canvas => {
        let link = document.createElement('a');
        link.download = 'diagrama.png';
        link.href = canvas.toDataURL();
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    });
}
```

5.4.2.11. uploadFile: Maneja la subida de archivos y envía el archivo al servidor.

- **Líneas 175-176**

Obtiene el archivo seleccionado.

- **Líneas 178-181**

Verifica si se ha seleccionado un archivo.

- **Líneas 183-185**

Crea un objeto FormData y añade el archivo y un parámetro adicional.

- **Líneas 187-195**

Envía una solicitud AJAX al servidor.

```
function uploadFile() {
    let input = $('#file')[0];
    let file = input.files[0];

    if (!file) {
        alert('Seleccione un archivo primero');
        return;
    }

    let formData = new FormData();
    formData.append('facturaPeritacion', file);
    formData.append('op', '2');

    $.ajax({
        type: 'POST',
        url: 'Default.aspx',
        data: formData,
        contentType: false,
        processData: false,
        success: handleSuccess,
        error: handleError
    });
}
```

5.4.2.12. generarDiagramaFlujo: Genera un diagrama de flujo utilizando GoJS.

- **Líneas 199-201**

Destruye el diagrama existente si existe.

- **Líneas 203-206**

Verifica si los datos del grafo son válidos.

- **Líneas 208-218**

Inicializa el diagrama de flujo.

- **Líneas 220-232**

Define una función para crear puertos de nodo.

- **Líneas 234-254**

Define la plantilla de nodo.

- **Líneas 256-261**

Define las formas de los nodos según el tipo.

- **Líneas 263-275**

Define la plantilla de enlace.

- **Línea 281**

Establece el modelo de datos del diagrama.

```
function generarDiagramaFlujo(GrafoData) {
    if (flujoDiagrama) {
        flujoDiagrama.div = null; // Destruir el diagrama existente
        flujoDiagrama = null;
    }
    if (!GrafoData || !GrafoData.Nodes || !GrafoData.Links) {
        console.error('Datos de grafo no válidos o faltantes.');
        return;
    }
    var $ = go.GraphObject.make;
    flujoDiagrama = $(go.Diagram, "grafoContainer", {
        "undoManager.isEnabled": true,
        layout: $(go.LayeredDigraphLayout, {
            direction: 0,
            layeringOption: go.LayeredDigraphLayout.LayerLongestPathSource,
            columnSpacing: 30,
            isRouting: true,
            packOption: go.LayeredDigraphLayout.PackMedian,
        })
    });
    function makePort(name, spot, output, input) {
        return $(go.Shape, "Circle",
            {
                fill: "transparent",
                stroke: null,
                desiredSize: new go.Size(8, 8),
                alignment: spot, alignmentFocus: spot,
                portId: name,
                fromSpot: spot, toSpot: spot,
                fromLinkable: output, toLinkable: input,
                cursor: "pointer"
            });
    }

    function createNodeTemplate(shapeType, fillColor) {
        return $(go.Node, "Spot",
            $(go.Panel, "Auto",
                $(go.Shape, shapeType, { fill: fillColor, stroke: "black" }),
                $(go.TextBlock, { margin: 5, editable: true, textAlign: "center", wrap:
go.TextBlock.WrapFit, width: 100 },
                    new go.Binding("text", "text", function (t) {
                        var words = t.split(" ");
                        if (words.length > 2) {
                            return words.slice(0, 2).join(" ") + "\n" + words.slice(2).join(" ");
                        } else {
                            return t;
                        }
                    })
                )
            ),
```

```
            makePort("T", go.Spot.Top, true, true),
            makePort("L", go.Spot.Left, true, true),
            makePort("R", go.Spot.Right, true, true),
            makePort("B", go.Spot.Bottom, true, true)
        );
    }

flujoDiagrama.nodeTemplateMap.add("StartEvent",
createNodeTemplate("Circle", "green"));
    flujoDiagrama.nodeTemplateMap.add("EndEvent",
createNodeTemplate("Circle", "red"));
    flujoDiagrama.nodeTemplateMap.add("Estandar",
createNodeTemplate("Rectangle", "white"));
    flujoDiagrama.nodeTemplateMap.add("Decisión",
createNodeTemplate("Diamond", "yellow"));
    flujoDiagrama.nodeTemplateMap.add("Aprobación",
createNodeTemplate("Diamond", "cyan"));
    flujoDiagrama.nodeTemplateMap.add("IntermediateEvent",
createNodeTemplate("Circle", "purple"));

    flujoDiagrama.linkTemplate =
        $(go.Link,
            {
                routing: go.Link.AvoidsNodes, corner: 5,
                relinkableFrom: true,
                relinkableTo: true,
                curviness: 20,
                adjusting: go.Link.End
            },
            $(go.Shape),
            $(go.Shape, { toArrow: "Standard" }),
            $(go.TextBlock, { margin: 3, editable: true }, new go.Binding("text",
"text"))
        );

    var model = $(go.GraphLinksModel);
    model.nodeDataArray = GrafoData.Nodes;
    model.linkDataArray = GrafoData.Links;

    flujoDiagrama.model = model;
}
```

5.4.2.13. generarDiagramaTortuga: Genera un diagrama de tortuga utilizando GoJS.

- **Líneas 284-288**
  Destruye el diagrama existente si existe.

- **Líneas 289-292**
  Verifica si los datos del grafo son válidos.

- **Líneas 293-300**
  Inicializa el diagrama de tortuga.

- **Líneas 303-310**
  Define varios degradados de color para los nodos.

- **Líneas 312-321**
  Define la plantilla de nodo.

- **Líneas 323-343**
  Define la plantilla de enlace.

- **Líneas 344-352**
  Define los nodos base del diagrama.

- **Líneas 354-376**
  Añade nodos para el diagrama si existen en los datos.

- **Líneas 378**
  Establece el modelo de datos del diagrama.

```
function generarDiagramaTortuga(datos) {
    if (tortugaDiagrama) {
        tortugaDiagrama.div = null;
        tortugaDiagrama = null;
    }
    if (!datos) {
        console.error('Datos de grafo no válidos o faltantes.');
        return;
    }
    var $ = go.GraphObject.make;

    tortugaDiagrama = $(go.Diagram, 'tortugaContainer', {
        layout: $(DoubleTreeLayout, {
            vertical: true,
            directionFunction: (node) => node.data.dir === 'down',
        }),
    });
```

```
tortugaDiagrama.nodeTemplate = $(go.Node,
        'Auto',
        { isShadowed: true },
        $(go.Shape,
            'RoundedRectangle',
            { fill: graygrad, stroke: '#D8D8D8', width: 150, height: 50 },
            new go.Binding('fill', 'color')
        ),
        $(go.TextBlock, { margin: 10, font: 'bold 14px Helvetica, bold Arial, sans-
serif' }, new go.Binding('text', 'key'))
    );
    tortugaDiagrama.linkTemplate = $(go.Link,
        { selectable: false },
        $(go.Shape, { strokeWidth: 2 },
            new go.Binding('stroke', '', (link) => {
                const toNode = link.toNode;
                if (toNode) {
                    if (toNode.data.key === 'Proceso') {
                        const parentNode = toNode.findTreeParentNode();
                        return parentNode.data.color;
                    } else {
                        return toNode.data.color;
                    }
                }
                return 'black';
            })
        )
    );
    const nodes = [
        { key: 'Proceso', color: whitegrad },
        { key: 'Fuentes', parent: 'Proceso', dir: 'up', color: bluegrad },
        { key: 'Entradas', parent: 'Proceso', dir: 'up', color: greengrad },
        { key: 'Recursos', parent: 'Proceso', dir: 'up', color: purplegrad },
        { key: 'Usuarios', parent: 'Proceso', dir: 'down', color: yellowgrad },
        { key: 'Salidas', parent: 'Proceso', dir: 'down', color: redgrad },
        { key: 'Receptores', parent: 'Proceso', dir: 'down', color: orangegrad }
    ];

    if (datos.Entradas) {
        for (const [key, value] of Object.entries(datos.Entradas)) {
            nodes.push({ key: value, parent: 'Entradas', dir: 'up', color: greengrad
});
            nodes.push({ key: key, parent: 'Fuentes', dir: 'up', color: bluegrad });
        }
    }
    for (const recurso of datos.Proceso.Recursos) {
        nodes.push({ key: recurso, parent: 'Recursos', dir: 'up', color: purplegrad
});
    }
```

```
    if (datos.Salidas) {
        for (const [key, value] of Object.entries(datos.Salidas)) {
            nodes.push({ key: value, parent: 'Receptores', dir: 'down', color:
orangegrad });
            nodes.push({ key: key, parent: 'Salidas', dir: 'down', color: redgrad });
        }
    }

    if (datos.Usuarios && Array.isArray(datos.Usuarios.Nombre)) {
        for (const usuario of datos.Usuarios.Nombre) {
            nodes.push({ key: usuario, parent: 'Usuarios', dir: 'down', color:
yellowgrad });
        }
    }

    tortugaDiagrama.model = new go.TreeModel(nodes);

}
```

5.4.3. Default.aspx.cs

5.4.3.1. Líneas 1-10

En este código vamos a usar varias bibliotecas para diferentes funcionalidades, a continuación, una breve explicación de cada una de ellas.

- **using System;**

Importa el espacio de nombres System que contiene clases fundamentales para .NET, como tipos básicos, manipulaciones de excepciones, etc.

- **using System.Collections.Generic;**

Importa el espacio de nombres que contiene interfaces y clases definidas para colecciones genéricas, como listas (List<T>), diccionarios (Dictionary<TKey, TValue>), etc.

- **using System.IO;**

Importa el espacio de nombres que permite trabajar con archivos y flujos de datos (lectura y escritura de archivos).

- **using System.Web.UI;**

Importa el espacio de nombres que proporciona clases e interfaces para crear y renderizar controles en una página web de ASP.NET.

- **using Newtonsoft.Json;**

Importa el espacio de nombres de la biblioteca JSON.NET (Newtonsoft.Json), que permite trabajar con JSON en .NET (serializar y deserializar objetos JSON).

- **using NPOI.SS.UserModel;**:

Importa el espacio de nombres de NPOI que define el modelo de usuario para trabajar con documentos de Excel de manera abstracta (interfaces comunes).

- **using NPOI.XSSF.UserModel;**:

Importa el espacio de nombres de NPOI que permite trabajar específicamente con archivos de Excel en formato XLSX (Excel 2007 y versiones posteriores).

- **using System.Web;**:

Importa el espacio de nombres que contiene clases e interfaces que habilitan el desarrollo de aplicaciones web ASP.NET.

- **using System.Text;**:

Importa el espacio de nombres que contiene clases para la manipulación de texto (por ejemplo, codificaciones, StringBuilder, etc.).

- **using System.Linq;**:

Importa el espacio de nombres que proporciona capacidades de consulta para colecciones de datos (LINQ: Language Integrated Query).

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Web.UI;
using Newtonsoft.Json;
using NPOI.SS.UserModel;
using NPOI.XSSF.UserModel;
using System.Web;
using System.Text;
using System.Linq;
```

5.4.3.2. Page_Load: Método que se ejecuta cuando la página se carga. Utiliza un switch para manejar diferentes operaciones basadas en un parámetro op en la solicitud.

```
// Este método se ejecuta cuando se carga la página
    protected void Page_Load(object sender, EventArgs e)
    {
        // Se usa una instrucción switch para evaluar el valor del parámetro
"op" en la solicitud (Request)
        switch (Request["op"])
        {
            // Si el valor del parámetro "op" es "1"
            case "1":
                // Llama al método EnviarFichero()
                EnviarFichero();
                break;
        }
    }
```

5.4.3.3. Líneas 29-108

Método privado que maneja la subida de archivos al servidor, procesa los datos del archivo Excel y genera una respuesta en formato JSON con varios conjuntos de datos. Además, crea diccionarios y tuplas para el manejo de la información. También crea una copia del archivo subido en el servidor.

```
        private void EnviarFichero()
        {
            // Inicializa una cadena vacía para la respuesta
            string Respuesta = string.Empty;

            // Obtiene los archivos subidos en la solicitud
            HttpFileCollection files = Request.Files;

            // Inicializa una cadena vacía para el nombre temporal del archivo
            string NombreTempFichero = string.Empty;

            // Obtiene el primer archivo de la colección de archivos
            HttpPostedFile file = files[0];

            // Crea un objeto anónimo para la respuesta con un estado inicial de
"ok"
            var respuesta = new { Estado = "ok" };

            // Verifica si hay al menos un archivo en la colección
            if (files.Count > 0)
            {
                try
                {
                    // Define la ruta donde se guardará el archivo en el servidor
                    string folderpath = Server.MapPath("~/Uploads/");
                    string filepath = folderpath +
Path.GetFileName(files[0].FileName);

                    // Si el directorio no existe, créalo
                    if (!Directory.Exists(folderpath))
                    {
                        Directory.CreateDirectory(folderpath);
                    }

                    // Guarda el archivo en la ruta especificada
                    files[0].SaveAs(filepath);
```

```
// Inicializa varios diccionarios para almacenar datos leídos del archivo Excel
                Dictionary<string, List<string>> datosTabla = new
Dictionary<string, List<string>>();
                Dictionary<string, string> entradas = new Dictionary<string,
string>();
                Dictionary<string, string> salidas = new Dictionary<string,
string>();
                Dictionary<string, List<string>> proceso = new Dictionary<string,
List<string>>();
                Dictionary<string, List<string>> indicadores = new
Dictionary<string, List<string>>();
                Dictionary<string, List<string>> usuarios = new Dictionary<string,
List<string>>();
                Dictionary<string, List<string>> procedimiento = new
Dictionary<string, List<string>>();

                // Lee la tabla Excel y llena el diccionario datosTabla
                LeerTablaExcel(filepath, datosTabla);

                // Limpia un diccionario global de identificadores a índices
                idenToIndex.Clear();

                // Crea un grafo a partir de los datos de Excel
                GrafoData grafo =
BpmnBuilder.CrearGrafoDesdeExcel(datosTabla);

                // Lee las entradas del archivo Excel y llena el diccionario
entradas
                entradas = LeerEntradas(filepath, entradas);

                // Lee las salidas del archivo Excel y llena el diccionario salidas
                salidas = LeerSalidas(filepath, salidas);

                // Lee el proceso del archivo Excel y obtiene el nombre del
proceso y el diccionario proceso
                var resultadoProceso = LeerProceso(filepath, proceso);
                string nombreProceso = resultadoProceso.Item1;
                proceso = resultadoProceso.Item2;

                // Lee los indicadores del archivo Excel y llena el diccionario de
indicadores
                indicadores = LeerIndicadores(filepath, indicadores);

                // Lee los usuarios del archivo Excel y llena el diccionario usuarios
                usuarios = LeerUsuarios(filepath, usuarios);
```

```
// Lee el procedimiento del archivo Excel y llena el diccionario procedimiento
                procedimiento = LeerProcedimiento(filepath, procedimiento);

                // Crea un objeto anónimo con todos los datos leídos y el estado
"ok"
                var jsonresult = new
                {
                    datosExcell = datosTabla,
                    Grafo = grafo,
                    Entradas = entradas,
                    Salidas = salidas,
                    Proceso = proceso,
                    Usuarios = usuarios,
                    Procedimiento = procedimiento,
                    nombreProceso = nombreProceso,
                    Estado = "ok"
                };

                // Inicializa una cadena vacía para la respuesta JSON
                string JsonRespuesta = string.Empty;

                // Serializa el objeto jsonresult a una cadena JSON con formato
indentado
                JsonRespuesta = JsonConvert.SerializeObject(jsonresult,
Formatting.Indented);

                // Establece el tipo de contenido de la respuesta HTTP a
"application/json"
                Response.ContentType = "application/json";

                // Escribe la cadena JSON en la respuesta HTTP
                Response.Write(JsonRespuesta);

                // Finaliza la respuesta HTTP
                try { Response.End(); }
                catch { }
            }
            catch (Exception ex)
            {
                // En caso de excepción, crea un objeto anónimo con el
mensaje de error
                var jsonresult = new
                {
                    error = ex.Message,
                    Estado = "nok"
                };
            }
        }
    }
```

5.4.3.4. Lineas 232-292

Método privado que lee un archivo Excel y carga los datos en un diccionario proporcionado como argumento.

```
private static void LeerTablaExcel(string filePath, Dictionary<string,
List<string>> datos)
      {
         // Cargar el archivo Excel usando un FileStream
         using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
         {
            // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
            IWorkbook workbook = new XSSFWorkbook(fileStream);

            // Obtener la primera hoja de trabajo (worksheet)
            ISheet worksheet = workbook.GetSheetAt(0);

            // Obtener el número total de filas físicas en la hoja de trabajo
            int totalFilas = worksheet.PhysicalNumberOfRows;

            // Obtener el número total de columnas físicas en la tercera fila
            int totalColumnas = worksheet.GetRow(2).PhysicalNumberOfCells;

            // Iterar sobre las columnas de la tercera fila (nombres de las
columnas)
            for (int col = 1; col < totalColumnas; col++)
            {
               // Obtener el nombre de la columna
               string nombreColumna =
worksheet.GetRow(2).GetCell(col).StringCellValue;

               // Verificar si el nombre de la columna es uno de los que nos
interesa
               if (nombreColumna == "Num. Identificador" || nombreColumna ==
"Actividad / Tarea" || nombreColumna == "Tipo" || nombreColumna ==
"Responsable" || nombreColumna == "Act. Predecesora" || nombreColumna
== "Act. Posterior" || nombreColumna == "Opción SI" || nombreColumna ==
"Opción NO")
               {
                  // Inicializar una lista vacía en el diccionario para esta columna
                  datos[nombreColumna] = new List<string>();
               }
            }
```

```
// Iterar sobre las filas comenzando desde la cuarta fila (índice 3)
            for (int fila = 3; fila < totalFilas; fila++)
            {
                // Iterar sobre las columnas comenzando desde la segunda
columna (índice 1)
                for (int col = 1; col < totalColumnas; col++)
                {
                    // Obtener el nombre de la columna y el valor de la celda actual
                    string nombreColumna =
worksheet.GetRow(2)?.GetCell(col)?.StringCellValue;
                    string ValorCelda =
worksheet.GetRow(fila)?.GetCell(col)?.ToString();

                    // Verificar si el nombre de la columna es uno de los que nos
interesa
                    if (nombreColumna == "Num. Identificador" || nombreColumna
== "Actividad / Tarea" || nombreColumna == "Tipo" || nombreColumna ==
"Responsable" || nombreColumna == "Act. Predecesora" || nombreColumna
== "Act. Posterior" || nombreColumna == "Opción SI" || nombreColumna ==
"Opción NO")
                    {
                        // Si el valor de la celda no es nulo ni vacío, agregarlo al
diccionario
                        if (!string.IsNullOrEmpty(ValorCelda))
                        {
                            datos[nombreColumna].Add(ValorCelda);
                        }
                        // Si la columna es "Actividad / Tarea" y el valor de la celda
es vacío, eliminar el identificador correspondiente
                        else if (nombreColumna == "Actividad / Tarea" &&
ValorCelda == "")
                        {
                            datos["Num.
Identificador"].Remove(worksheet.GetRow(fila).GetCell(col - 1).ToString());
                            fila++;
                            col = 0; // Reiniciar la columna para la siguiente fila
                        }
                    }
                }
            }
        }
```

5.4.3.5. Líneas 293-315

Método estático que lee las entradas desde una hoja específica del archivo Excel y las almacena en un diccionario.

```
        private static Dictionary<string, string> LeerEntradas(string filePath,
Dictionary<string, string> entradas)
        {
            // Cargar el archivo Excel usando un FileStream
            using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
            {
                // Crear un objeto IWorkbook a partir del FileStream (asumiendo que
es un archivo .xlsx)
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la cuarta hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(3); // Índice 3
corresponde a la cuarta hoja

                // Obtener el número total de filas físicas en la hoja de trabajo
                int totalFilas = worksheet.PhysicalNumberOfRows;

                // Definir la columna inicial que se va a leer
                int col = 1;

                // Iterar sobre las filas comenzando desde la quinta fila (índice 4)
                for (int fil = 4; fil < totalFilas; fil++)
                {
                    // Obtener los valores de las celdas de la columna 'col' y 'col + 1'
                    string Fuente =
worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    string Entrada = worksheet.GetRow(fil).GetCell(col +
1).StringCellValue;

                    // Verificar si ambos valores no son vacíos
                    if (!string.IsNullOrEmpty(Fuente) &&
!string.IsNullOrEmpty(Entrada))
                    {
                        // Agregar la entrada al diccionario
                        entradas.Add(Fuente, Entrada);
                    }
                }
            }

            // Devolver el diccionario con las entradas leídas
            return entradas;
        }
```

5.4.3.6. Líneas 316-338

Método estático que lee las salidas desde una hoja específica del archivo Excel y las almacena en un diccionario.

```
      private static Dictionary<string, string> LeerSalidas(string filePath,
Dictionary<string, string> salidas)
      {
         // Cargar el archivo Excel usando un FileStream
         using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
         {
            // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
            IWorkbook workbook = new XSSFWorkbook(fileStream);

            // Obtener la cuarta hoja de trabajo (worksheet)
            ISheet worksheet = workbook.GetSheetAt(3); // Índice 3
corresponde a la cuarta hoja

            // Obtener el número total de filas físicas en la hoja de trabajo
            int totalFilas = worksheet.PhysicalNumberOfRows;

            // Definir la columna inicial que se va a leer
            int col = 4;

            // Iterar sobre las filas comenzando desde la quinta fila (índice 4)
            for (int fil = 4; fil < totalFilas; fil++)
            {
               // Obtener los valores de las celdas de la columna 'col' y 'col + 1'
               string Salidas =
worksheet.GetRow(fil).GetCell(col).StringCellValue;
               string Receptores = worksheet.GetRow(fil).GetCell(col +
1).StringCellValue;

               // Verificar si ambos valores no son vacíos
               if (!string.IsNullOrEmpty(Salidas) &&
!string.IsNullOrEmpty(Receptores))
               {
                  // Agregar la salida al diccionario
                  salidas.Add(Salidas, Receptores);
               }
            }
         }

         // Devolver el diccionario con las salidas leídas
         return salidas;
      }
```

5.4.3.7. Lineas 339-380

Método estático que lee el proceso desde una hoja específica del archivo Excel y devuelve el nombre del proceso y los detalles en un diccionario.

```
        private static Tuple<string, Dictionary<string, List<string>>>
LeerProceso(string filePath, Dictionary<string, List<string>> proceso)
        {
            // Inicializar una variable para el nombre del proceso
            string nombreProceso = string.Empty;

            // Cargar el archivo Excel usando un FileStream
            using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
            {
                // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la tercera hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(2); // Índice 2
corresponde a la tercera hoja

                // Obtener el número total de filas físicas en la hoja de trabajo
                int totalFilas = worksheet.PhysicalNumberOfRows;

                // Definir la columna inicial que se va a leer
                int col = 1;

                // Leer el nombre del proceso de la celda "C3" (fila 2, columna 2)
                nombreProceso =
worksheet.GetRow(2).GetCell(2).StringCellValue;

                // Leer datos de las filas 2 a 4 y agregarlos al diccionario 'proceso'
                for (int fil = 2; fil < 5; fil++)
                {
                    string dato =
worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    string dato2 = worksheet.GetRow(fil).GetCell(2).StringCellValue;
                    List<string> detalles = new List<string> { dato2 };
                    proceso.Add(dato, detalles);
                }
                // Leer y agregar datos de la fila 6 y 7 al diccionario 'proceso'
                string datoFila6 =
worksheet.GetRow(6).GetCell(1).StringCellValue;
                string datoFila7 =
worksheet.GetRow(7).GetCell(1).StringCellValue;
                proceso.Add(datoFila6, new List<string> { datoFila7 });
```

```
// Leer datos a partir de la fila 10 y agregarlos al diccionario 'proceso'
            string datoFila10 = worksheet.GetRow(10).GetCell(1).StringCellValue;
            List<string> ultimosDetalles = new List<string>();
            for (int fil = 11; fil < totalFilas; fil++)
            {
                string datoUltimaFila =
worksheet.GetRow(fil).GetCell(1).StringCellValue;
                ultimosDetalles.Add(datoUltimaFila);
            }
            proceso.Add(datoFila10, ultimosDetalles);

            // Retornar una tupla con el nombre del proceso y el diccionario 'proceso'
            return Tuple.Create(nombreProceso, proceso);
        }
    }
```

5.4.3.8. Líneas 381-426

Método estático que lee los indicadores desde una hoja específica del archivo Excel y los almacena en un diccionario

```
        private static Dictionary<string, List<string>> LeerIndicadores(string
filePath, Dictionary<string, List<string>> indicadores)
        {
            // Cargar el archivo Excel usando un FileStream
            using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
            {
                // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la quinta hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(4); // Índice 4
corresponde a la quinta hoja

                // Obtener el número total de filas físicas en la hoja de trabajo
                int totalFilas = worksheet.PhysicalNumberOfRows;

                // Iterar sobre las columnas desde la segunda columna hasta la
sexta (índices 1 a 5)
                for (int col = 1; col < 6; col++)
                {
                    // Obtener el nombre de la columna desde la fila 1
(encabezado)
                    string nombreColumna =
worksheet.GetRow(1).GetCell(col).StringCellValue;

                    // Crear una nueva lista para almacenar los detalles de cada
columna
                    List<string> detalles = new List<string>();

                    // Iterar sobre las filas comenzando desde la tercera fila (índice
2)
                    for (int fil = 2; fil < totalFilas; fil++)
                    {
                        // Obtener la celda actual
                        var cell = worksheet.GetRow(fil).GetCell(col);
                        string dato;

                        if (cell == null)
                        {
                            // No hacer nada si la celda está vacía
                            continue;
                        }
```

```
else if (cell.CellType == CellType.Numeric)
                    {
                        // Convertir el valor numérico a string y agregarlo a la
lista de detalles
                        dato = cell.NumericCellValue.ToString();
                        detalles.Add(dato);
                    }
                    else if (cell.StringCellValue == "" || cell.StringCellValue
== "Nota: Se puede definir un máximo de 4 indicadores por año, uno
de cada tipo (U.medida), para cada proceso.")
                    {
                        // No hacer nada si el valor de la celda es vacío o
igual al mensaje específico
                        continue;
                    }
                    else
                    {
                        // Si no es numérico, obtener el valor como string
directamente y agregarlo a la lista de detalles
                        dato = cell.StringCellValue;
                        detalles.Add(dato);
                    }
                }

                // Agregar la lista de detalles al diccionario con el nombre
de la columna como clave
                indicadores.Add(nombreColumna, detalles);
            }
        }

        // Devolver el diccionario con los indicadores leídos
        return indicadores;
    }
```

5.4.3.9. Líneas 427-460

Método estático que lee los usuarios desde una hoja específica del archivo Excel y los almacena en un diccionario.

```
        private static Dictionary<string, List<string>> LeerUsuarios(string
filePath, Dictionary<string, List<string>> usuarios)
        {
            // Cargar el archivo Excel usando un FileStream
            using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
            {
                // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la sexta hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(5); // Índice 5
corresponde a la sexta hoja

                // Obtener el número total de filas físicas en la hoja de trabajo
                int totalFilas = worksheet.PhysicalNumberOfRows;

                // Iterar sobre las columnas desde la segunda columna hasta la
cuarta (índices 1 a 3)
                for (int col = 1; col < 4; col++)
                {
                    // Obtener el nombre de la columna desde la fila 1
(encabezado)
                    string nombreColumna =
worksheet.GetRow(1).GetCell(col).StringCellValue;

                    // Crear una nueva lista para almacenar los detalles de cada
columna
                    List<string> detalles = new List<string>();

  // Iterar sobre las filas comenzando desde la tercera fila (índice 2)
                    for (int fil = 2; fil <= totalFilas; fil++)
                    {
                        // Obtener el valor de la celda actual como string
                        string cell =
worksheet.GetRow(fil).GetCell(col).StringCellValue;
                        if (cell == "")
                        {
                            // No hacer nada si la celda está vacía
                            continue;
                        }
```

```
else
                    {
                        // Agregar el valor de la celda a la lista de detalles
                        detalles.Add(cell);
                    }
                }

                // Agregar la lista de detalles al diccionario con el nombre de la
columna como clave
                usuarios.Add(nombreColumna, detalles);
            }
        }

        // Devolver el diccionario con los usuarios leídos
        return usuarios;
    }
```

5.4.3.10. Lineas 461-514

Método estático que lee los procedimientos desde una hoja específica del archivo Excel y los almacena en un diccionario.

```
        private static Dictionary<string, List<string>> LeerProcedimiento(string
filePath, Dictionary<string, List<string>> datos)
        {
            // Cargar el archivo Excel usando un FileStream
            using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
            {
                // Crear un objeto IWorkbook a partir del FileStream (asumiendo
que es un archivo .xlsx)
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la primera hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(0); // Índice 0
corresponde a la primera hoja

                // Obtener el número total de filas físicas y columnas en la hoja de
trabajo
                int totalFilas = worksheet.PhysicalNumberOfRows;
                int totalColumnas = worksheet.GetRow(2).PhysicalNumberOfCells;

                // Iterar sobre las columnas de la tercera fila (nombres de las
columnas)
                for (int col = 1; col <= totalColumnas; col++)
                {
                    // Obtener el nombre de la columna desde la fila 3 (tercera fila)
                    string nombreColumna =
worksheet.GetRow(2).GetCell(col).StringCellValue;

                    // Inicializar una nueva lista para almacenar los valores de esta
columna
                    datos[nombreColumna] = new List<string>();
                }

                // Iterar sobre las filas comenzando desde la cuarta fila (índice 3)
                for (int fila = 3; fila <= totalFilas; fila++)
                {
                    // Iterar sobre las columnas comenzando desde la primera
columna (índice 1)
                    for (int col = 1; col <= totalColumnas; col++)
                    {
                        // Obtener el nombre de la columna para la celda actual
                        string nombreColumna =
worksheet.GetRow(2)?.GetCell(col).StringCellValue;
```

```
                    // Obtener el valor de la celda como string (incluye manejo
para celdas vacías o nulas)
                    string ValorCelda =
worksheet.GetRow(fila)?.GetCell(col)?.ToString();

                    if (!string.IsNullOrEmpty(ValorCelda))
                    {
                        // Agregar el valor al diccionario bajo el nombre de la
columna correspondiente
                        datos[nombreColumna].Add(ValorCelda);
                    }
                    else if (nombreColumna == "Actividad / Tarea" &&
string.IsNullOrEmpty(ValorCelda))
                    {
                        // Manejo especial cuando la columna "Actividad / Tarea"
está vacía
                        datos["Num.
Identificador"].Remove(worksheet.GetRow(fila).GetCell(col - 1).ToString());
                        fila++;
                        col = 0;
                    }
                    // No se hace nada si ValorCelda es null
                }
            }

            // Devolver el diccionario actualizado con los datos del
procedimiento
            return datos;
        }
    }
```

5.4.3.11. Líneas 109-113

Propiedades:

- **Nodes**: Lista de nodos para representar en un diagrama.
- **Links**: Lista de enlaces entre nodos para representar en un diagrama.

```
public class GrafoData
    {
        public List<GoJsNode> Nodes { get; set; } = new
List<GoJsNode>();
        public List<GoJsLink> Links { get; set; } = new List<GoJsLink>();
    }
```

5.4.3.12. Líneas 115-121

Propiedades:

- **key**: Clave única del nodo.
- **text**: Texto descriptivo del nodo.
- **category**: Categoría del nodo para propósitos de representación visual.
- **predecessors**: Lista de predecesores del nodo en el grafo.

```
public class GoJsNode
    {
        public string key { get; set; }
        public string text { get; set; }
        public string category { get; set; }
        public List<string> predecessors { get; set; } = new List<string>();
    }
```

5.4.3.13. Clase GoJSLink

Propiedades:

- **key**: Clave única del enlace.
- **from**: Clave del nodo origen del enlace.
- **to**: Clave del nodo destino del enlace.
- **text**: Texto descriptivo del enlace.

```
public class GoJsLink
{
   public string key { get; set; }
   public string from { get; set; }
   public string to { get; set; }
   public string text { get; set; }
}
```

5.4.3.14. Líneas 131-218

Método estático que construye un grafo (GrafoData) basado en los datos de una tabla Excel proporcionada como argumento.

```
public static class BpmnBuilder
{
   public static GrafoData CrearGrafoDesdeExcel(Dictionary<string,
List<string>> datosTabla)
   {
      var grafo = new GrafoData(); // Instancia del objeto GrafoData
que almacenará nodos y enlaces
      int totalFilas = datosTabla.First().Value.Count; // Número total
de filas en los datos de la tabla

      // Creación de nodos de inicio y fin estáticos y agregados al
grafo
      var nodoInicio = new GoJsNode { key = "inicio", text = "Inicio",
category = "StartEvent" };
      var nodoFin = new GoJsNode { key = "fin", text = "Fin",
category = "EndEvent" };
      grafo.Nodes.Add(nodoInicio);
      grafo.Nodes.Add(nodoFin);

      // Iteración sobre cada fila de los datos de la tabla
      for (int fila = 0; fila < totalFilas; fila++)
      {
```

```
var nodo = new GoJsNode(); // Nuevo nodo para cada fila
                string posteriores = null; // Variable para almacenar los nodos
posteriores

                // Iteración sobre cada columna de la fila actual
                foreach (var kvp in datosTabla)
                {
                    string nombreColumna = kvp.Key; // Nombre de la columna
actual
                    string valorCelda = kvp.Value[fila]; // Valor de la celda actual en
la fila correspondiente

                    // Asignación de propiedades del nodo según la columna actual
                    if (nombreColumna == "Num. Identificador")
                        nodo.key = valorCelda; // Asignar clave del nodo
                    else if (nombreColumna == "Actividad / Tarea")
                        nodo.text = valorCelda; // Asignar texto del nodo
                    else if (nombreColumna == "Tipo")
                        nodo.category = valorCelda; // Asignar categoría del nodo
                    else if (nombreColumna == "Act. Posterior" &&
!string.IsNullOrEmpty(valorCelda))
                    {
                        posteriores = valorCelda; // Almacenar nodos posteriores si la
columna es "Act. Posterior"
                    }
                    else if (nombreColumna == "Act. Posterior" && valorCelda ==
"fin")
                    {
                        // Si el nodo posterior es "fin", agregar enlace al nodo de fin
                        grafo.Links.Add(new GoJsLink { key = "fin" + nodo.key, from =
nodo.key, to = "fin", text = "" });
                    }
                }
                 // Manejo de nodos intermedios y enlaces
            if (!string.IsNullOrEmpty(posteriores))
            {
                        // Verificar si hay un nodo previo para potencial nodo
                 intermedio
                if (fila > 0)
                {
                var nodoPrevio = grafo.Nodes[grafo.Nodes.Count - 1]; // Obtener el
nodo previo
                        var posterioresPrevios = datosTabla["Act. Posterior"][fila -
                 1]; // Obtener posteriores del nodo previo
```

```
// Crear nodo intermedio si los posteriores son los mismos que los del nodo previo
                    if (posteriores == posterioresPrevios)
                    {
                        var nodoIntermedio = new GoJsNode
                        {
                            key = nodo.key + "_intermedio",
                            text = "",
                            category = "IntermediateEvent"
                        };
                        grafo.Nodes.Add(nodoIntermedio); // Agregar nodo intermedio al grafo

                        // Conectar nodos originales al nodo intermedio
                        grafo.Links.Add(new GoJsLink { key = "link_" + nodoPrevio.key + "_" + nodoIntermedio.key, from = nodoPrevio.key, to = nodoIntermedio.key, text = "" });
                        grafo.Links.Add(new GoJsLink { key = "link_" + nodo.key + "_" + nodoIntermedio.key, from = nodo.key, to = nodoIntermedio.key, text = "" });

                        // Eliminar conexiones originales de los nodos a los posteriores
                        grafo.Links.RemoveAll(link => link.from == nodoPrevio.key && posteriores.Split(';').Contains(link.to));
                        grafo.Links.RemoveAll(link => link.from == nodo.key && posteriores.Split(';').Contains(link.to));

                        // Conectar el nodo intermedio a los posteriores
                        foreach (var successor in posteriores.Split(';'))
                        {
                            grafo.Links.Add(new GoJsLink { key = "link_" + nodoIntermedio.key + "_" + successor, from = nodoIntermedio.key, to = successor, text = "" });
                        }

                        grafo.Nodes.Add(nodo); // Agregar el nodo actual al grafo

                        continue; // Evitar agregar enlaces adicionales para este nodo en esta iteración
                    }
                }
                // Agregar enlaces regulares para nodos no intermedios
                foreach (var successor in posteriores.Split(';'))
                {
                    grafo.Links.Add(new GoJsLink { key = "link_" + nodo.key + "_" + successor, from = nodo.key, to = successor, text = "" });
                }
            }
```

```
grafo.Nodes.Add(nodo); // Agregar el nodo actual al grafo
                AddLinks(grafo, nodo, fila); // Agregar enlaces adicionales basados en sucesores
            }

            // Conectar el nodo de inicio con el primer nodo en la lista de nodos
            grafo.Links.Add(new GoJsLink { key = "enlace_inicio", from = nodoInicio.key, to = grafo.Nodes[2].key, text = "" });

            return grafo; // Devolver el grafo construido
        }

        // Método para agregar enlaces basados en sucesores de un nodo
        private static void AddLinks(GrafoData grafo, GoJsNode nodo, int fila)
        {
            // No se implementó esta función en la conversación anterior, se debe agregar aquí si es necesario
        }
    }
```

6. Resultados

A continuación, se mostrará un ejemplo de uso de la aplicación, mostrando su funcionamiento y resultado obtenido.

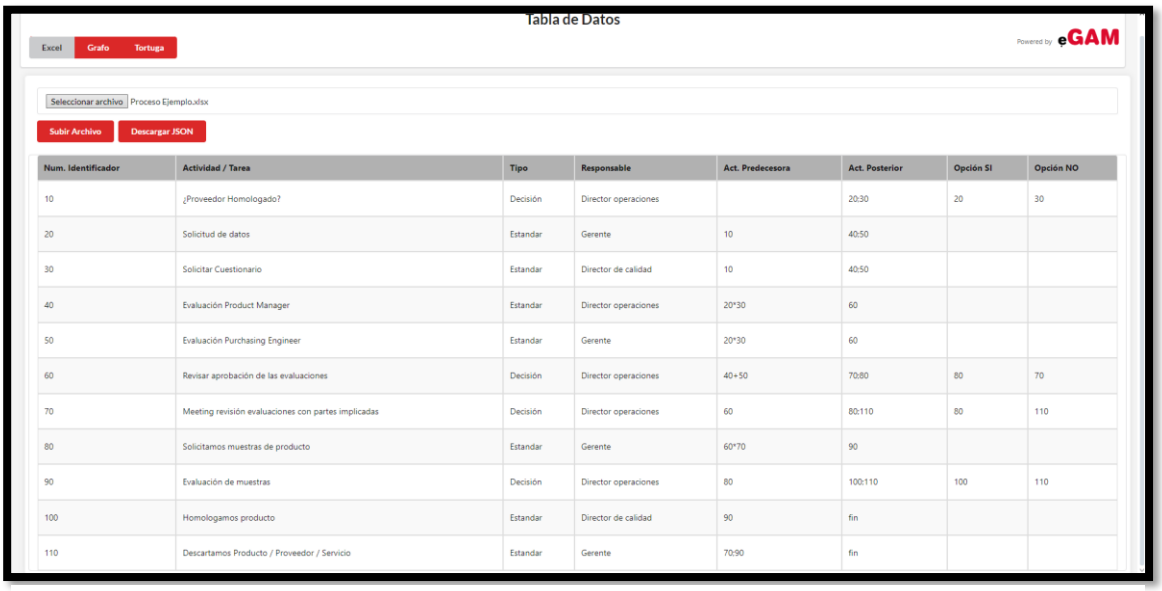
Inicialmente, la página muestra todos los botones ocultos o deshabilitados, excepto el botón para seleccionar y subir archivos.

Imagen proyecto 1

Después de seleccionar un archivo, su nombre aparecerá en el cuadro designado.

Imagen proyecto 2

Al pulsar el botón de subir archivo, se mostrará una versión reducida de la primera hoja del documento Excel subido. En esta hoja se pueden ver valores como el número identificador de la tarea, el nombre de las tareas, su tipo (que puede ser decisión, aprobación o estándar), las actividades predecesoras y posteriores de cada tarea, y dos columnas adicionales que indican, en caso de ser tareas de tipo aprobación o decisión, cuáles son sus tareas de destino según si han sido aprobadas o no.



Num. Identificador	Actividad / Tarea	Tipo	Responsable	Act. Predecesora	Act. Posterior	Opción SI	Opción NO
10	¿Proveedor Homologado?	Decisión	Director operaciones		20;30	20	30
20	Solicitud de datos	Estandar	Gerente	10	40;50		
30	Solicitar Cuestionario	Estandar	Director de calidad	10	40;50		
40	Evaluación Product Manager	Estandar	Director operaciones	20*30	60		
50	Evaluación Purchasing Engineer	Estandar	Gerente	20*30	60		
60	Revisar aprobación de las evaluaciones	Decisión	Director operaciones	40+50	70;80	80	70
70	Meeting revisión evaluaciones con partes implicadas	Decisión	Director operaciones	60	80;110	80	110
80	Solicitamos muestras de producto	Estandar	Gerente	60*70	90		
90	Evaluación de muestras	Decisión	Director operaciones	80	100;110	100	110
100	Homologamos producto	Estandar	Director de calidad	90	fin		
110	Descartamos Producto / Proveedor / Servicio	Estandar	Gerente	70;90	fin		

Imagen proyecto 3

Después de subir el archivo, se activará el botón "Grafo". Al pulsarlo, la vista cambiará de la tabla anterior a un diagrama de flujo que muestra de forma visual el orden de ejecución de las tareas, permitiendo identificar cada tarea y su tipo de un simple vistazo.

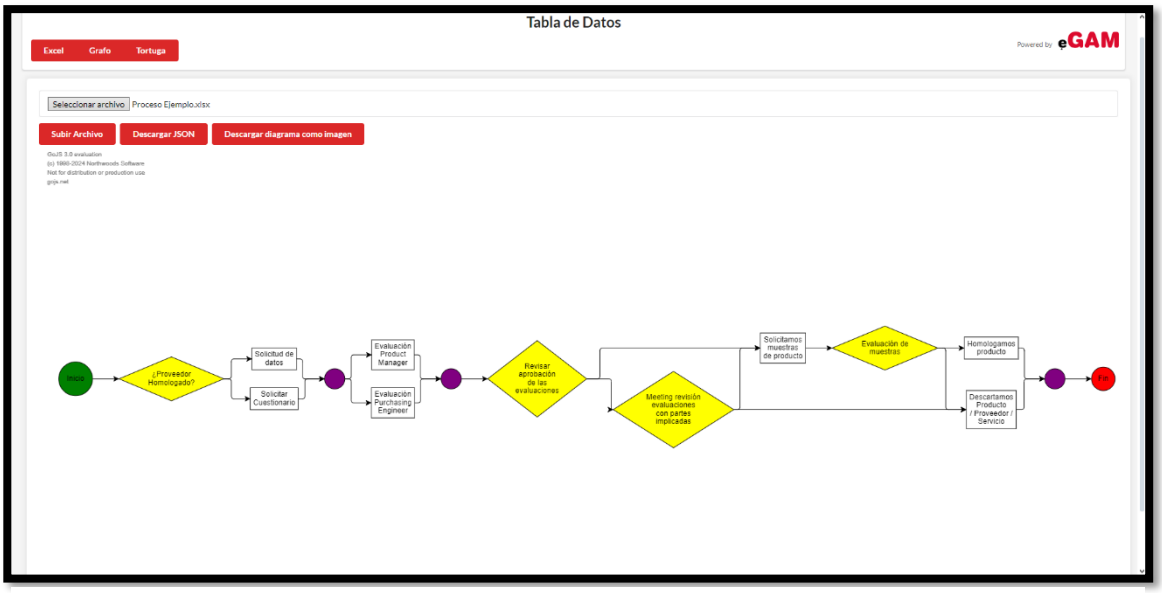


Imagen proyecto 4

Del mismo modo, el botón "Tortuga" se activará. Al pulsarlo, la vista cambiará de la tabla anterior a un diagrama de tortuga que presenta de forma visual información del proceso, como las entradas y salidas, los recursos utilizados, etc., permitiendo identificar cada tarea y su tipo fácilmente.

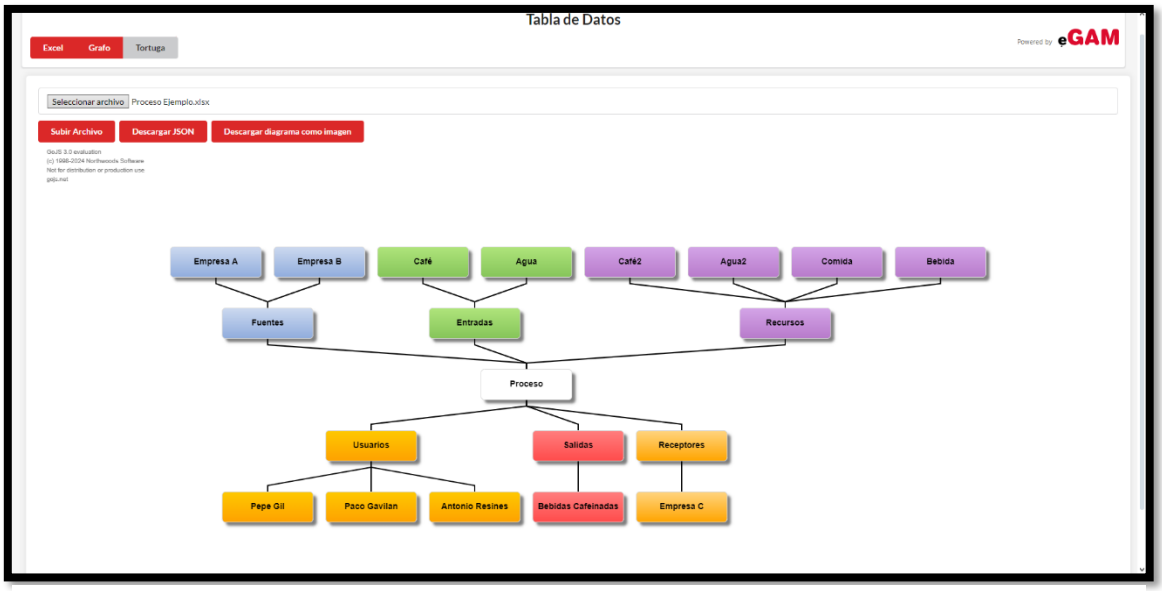
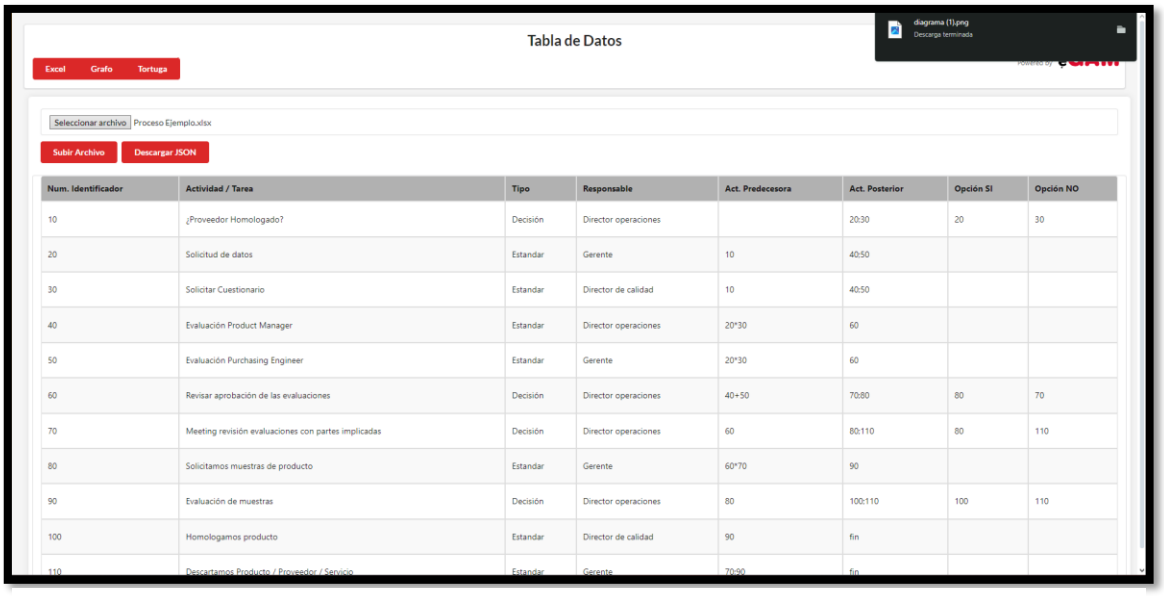


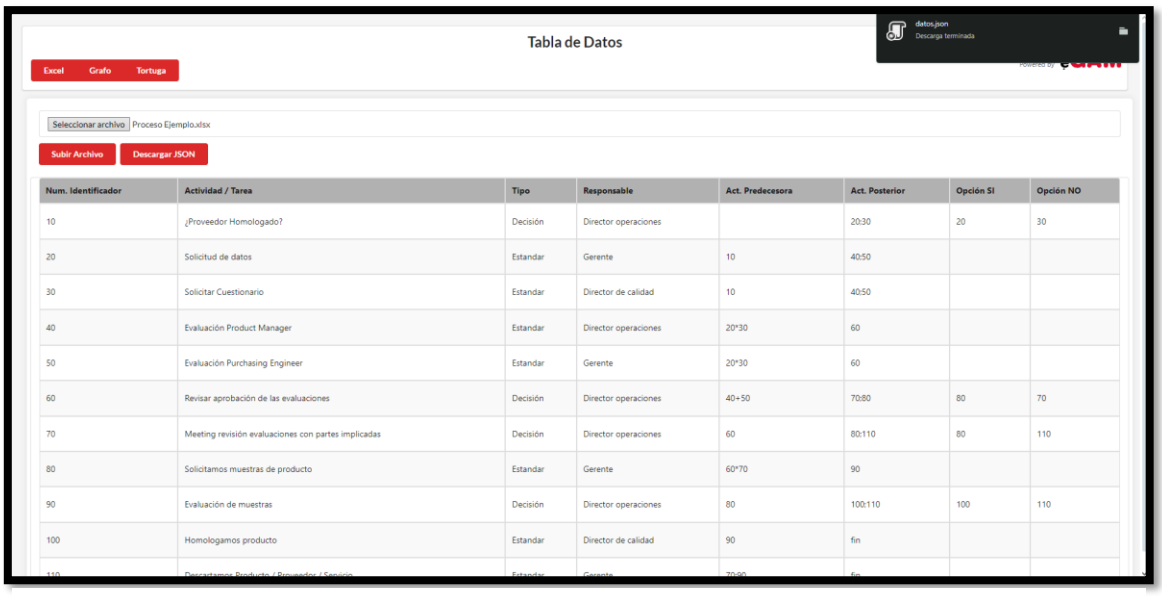
Imagen proyecto 5

Los botones "Descargar JSON" y "Descargar Diagrama como Imagen" funcionarán y permitirán iniciar las descargas correspondientes



Num. Identificador	Actividad / Tarea	Tipo	Responsable	Act. Predecesora	Act. Posterior	Opción SI	Opción NO
10	¿Proveedor Homologado?	Decisión	Director operaciones		20;30	20	30
20	Solicitud de datos	Estandar	Gerente	10	40;50		
30	Solicitar Cuestionario	Estandar	Director de calidad	10	40;50		
40	Evaluación Product Manager	Estandar	Director operaciones	20*30	60		
50	Evaluación Purchasing Engineer	Estandar	Gerente	20*30	60		
60	Revisar aprobación de las evaluaciones	Decisión	Director operaciones	40+50	70;80	80	70
70	Meeting revisión evaluaciones con partes implicadas	Decisión	Director operaciones	60	80;110	80	110
80	Solicitamos muestras de producto	Estandar	Gerente	60*70	90		
90	Evaluación de muestras	Decisión	Director operaciones	80	100;110	100	110
100	Homologamos producto	Estandar	Director de calidad	90	fin		
110	Descartamos Producto / Proveedor / Servicio	Estandar	Gerente	70;90	fin		

Imagen proyecto 6



Num. Identificador	Actividad / Tarea	Tipo	Responsable	Act. Predecesora	Act. Posterior	Opción SI	Opción NO
10	¿Proveedor Homologado?	Decisión	Director operaciones		20;30	20	30
20	Solicitud de datos	Estandar	Gerente	10	40;50		
30	Solicitar Cuestionario	Estandar	Director de calidad	10	40;50		
40	Evaluación Product Manager	Estandar	Director operaciones	20*30	60		
50	Evaluación Purchasing Engineer	Estandar	Gerente	20*30	60		
60	Revisar aprobación de las evaluaciones	Decisión	Director operaciones	40+50	70;80	80	70
70	Meeting revisión evaluaciones con partes implicadas	Decisión	Director operaciones	60	80;110	80	110
80	Solicitamos muestras de producto	Estandar	Gerente	60*70	90		
90	Evaluación de muestras	Decisión	Director operaciones	80	100;110	100	110
100	Homologamos producto	Estandar	Director de calidad	90	fin		
110	Descartamos Producto / Proveedor / Servicio	Estandar	Gerente	70;90	fin		

Imagen proyecto 7

7. Conclusiones

En general, el programa cumple con todos los objetivos planteados desde el principio. Además, el desarrollo ha sido acorde a lo previamente acordado con la empresa.

Si bien existe la posibilidad de mejora mediante la implementación de otros formatos de diagramas para una mayor visualización de datos, esta aplicación es suficiente para realizar análisis generales.

La obtención del diagrama de flujo es especialmente efectiva debido a su adaptabilidad.

La representación del diagrama de tortuga podría haber sido mejor con un mayor entendimiento del funcionamiento de los “layouts” de la librería GoJS.

Cabe destacar el aprendizaje de nuevas tecnologías necesario para este trabajo, como la interacción con servicios web y el uso de API de servicios privados que requieren autenticaciones y permisos para realizar las peticiones. Estos métodos aprendidos, junto con la estructuración de un programa modular y el diseño de estructuras de datos reutilizables, serán de gran utilidad en el futuro.

Es importante mencionar que este proyecto es el primero al que el autor se enfrenta en solitario con un alcance y dimensiones tan amplios. El acercamiento a este proyecto y la metodología aprendida han resultado interesantes y serán útiles en el futuro.

En general, creo que el programa tiene margen de mejora y varias cosas se han quedado pendientes, mayormente por falta de tiempo. Sin embargo, considero que el resultado final es una buena primera versión de este proyecto y una valiosa adición a la plataforma de análisis que se encuentra en construcción.

7.1. Mejoras futuras

Actualmente, el diagrama de flujo cumple su función principal de mostrar el orden de ejecución de las tareas de manera efectiva. Sin embargo, hay margen para mejorar su presentación y funcionalidad.

- **Personalización de Estilos:** Incorporar opciones para personalizar los colores, formas, y tamaños de los nodos y enlaces en el diagrama. Esto permitirá a los usuarios adaptar el diagrama a las necesidades y preferencias específicas de su organización.

- **Interactividad Mejorada:** Añadir funcionalidades interactivas como el zoom, la posibilidad de arrastrar y soltar nodos para reorganizarlos, y la visualización de detalles adicionales al hacer clic en un nodo.
- **Automatización del Formato:** Implementar algoritmos que optimicen automáticamente el layout del diagrama para evitar la superposición de nodos y enlaces, asegurando una visualización clara y ordenada.

La apariencia visual de la aplicación juega un papel crucial en la experiencia del usuario. Mejorar la estética general no solo hará la aplicación más agradable visualmente, sino que también puede mejorar su usabilidad.

- **Diseño Moderno y Atractivo:** Revisar y actualizar la interfaz de usuario (UI) utilizando principios de diseño moderno, como el uso de colores suaves, iconografía clara y tipografías legibles.
- **Tema Oscuro y Claro:** Implementar opciones de temas oscuro y claro para adaptarse a las preferencias de los usuarios y mejorar la accesibilidad en diferentes condiciones de iluminación.
- **Consistencia Visual:** Asegurar que todos los componentes de la UI tengan un diseño consistente. Esto incluye botones, formularios, tablas y gráficos.
- **Feedback Visual:** Proveer feedback visual inmediato al usuario para acciones como cargar un archivo, realizar una búsqueda o cambiar vistas. Esto puede incluir animaciones sutiles y mensajes de estado.

Generar informes explicativos automáticos añadirá un valor significativo a la aplicación, facilitando la comprensión y comunicación de los análisis realizados.

- **Generación Automática de Informes:** Desarrollar una funcionalidad que permita la generación automática de informes detallados en formato PDF o Word, incluyendo el diagrama de flujo, el diagrama de tortuga y los datos relevantes.
- **Contenido del Informe:** Los informes deberían incluir una descripción de cada tarea, sus relaciones con otras tareas, los resultados de las decisiones y aprobaciones, y cualquier otro dato pertinente.
- **Personalización de Informes:** Permitir a los usuarios personalizar el contenido del informe, seleccionando qué secciones y detalles incluir, así como la posibilidad de añadir comentarios y observaciones personales.
- **Plantillas Predefinidas:** Ofrecer varias plantillas predefinidas para informes, adaptadas a diferentes tipos de análisis y audiencias, como informes ejecutivos, técnicos y de auditoría.

Implementar estas mejoras no solo incrementará la funcionalidad y usabilidad de la aplicación, sino que también elevará su valor para los usuarios, facilitando una mejor comprensión y gestión de los procesos empresariales.

8. Bibliografía

Bibliografía

Safety Culture. (15 de enero de 2024). Obtenido de https://safetyculture.com/es/temas/diagrama-de-tortuga/

Mermaid.js.org. (14 de marzo de 2024). Obtenido de https://github.com/mermaid-js/mermaid

GoJS. (15 de marzo de 2024). Obtenido de https://gojs.net/latest/index.html

JointJS (15 de marzo de 2024). Obtenido de https://www.jointjs.com

Microsoft Power Bi. (25 de abril de 2024). Obtenido de https://www.microsoft.com/es-es/power-platform/products/power-bi

Microsoft Visio. (24 de mayo de 2024). Obtenido de https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software

Apache Airflow (5 de abril de 2024). Obtenido de https://airflow.apache.org

Tableau (10 de junio de 2024). Obtenido de https://www.tableau.com/en-gb/trial/tableau-software

Semantic ui (20 de febrero de 2024). Obtenido de https://semantic-ui.com

9. Anexo

9.1. Código

Default.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ABDFinal.WebForm1" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="Script/jquery-3.7.1.min.js"></script>
    <script src="https://unpkg.com/gojs/release/go-debug.js"></script>
    <script src="Script/semantic.min.js"></script>
    <script src="Script/DoubleTreeLayout.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js"></
script>
    <script
src="Script/Funciones.js?t=<%=DateTime.Now.ToString("yyyyMMddHHmmss_fffffff"
) %>"></script>
    <link href="CSS/semantic.min.css" rel="stylesheet" />
    <title>Data Table</title>
    <style>
        body {
            background-color: #f5f5f5;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            min-height: 100vh;
        }

        #content {
            flex: 1;
            padding: 20px;
        }

        form.ui.form {
            background-color: #fff;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            max-width: 100%;
            margin: auto;
        }
```

```
.ui.red.primary.submit.button {
    background-color: #db2828;
    color: #fff;
    margin-top: 10px;
}

#excelContainer {
    margin-top: 20px;
}

.ui.segment {
    position: relative;
}

    .ui.segment h2.ui.header {
        text-align: center;
    }

table.ui.striped.table th,
table.ui.striped.table td {
    border: 1px solid #ddd;
    padding: 10px;
    text-align: left;
}

table.ui.striped.table th {
    background-color: #8080809c;
}

table.ui.striped.table tbody tr:nth-child(even) {
    background-color: #f9f9f9;
}

table.ui.striped.table tbody tr:hover {
    background-color: #f1f1f1;
}

.cls-1, .cls-2, .cls-4 {
    fill-rule: evenodd;
}

.cls-2 {
    fill: #DC0A2D;
}

.cls-3 {
    fill: #202020;
    stroke: #000;
    stroke-width: 1px;
}

.cls-4 {
    fill: #0D0D0D;
}
```

```
        .logo-egam {
            position: absolute;
            top: 10px;
            right: 10px;
            display: flex;
            align-items: center;
        }

        .power-by {
            top: 5px;
            font-size: 0.8em;
            color: #666;
            margin-right: 10px;
        }

        .capa-svg {
            font-size: 0.8em;
            color: #666;
        }

        button[disabled], .ui.button[disabled], .ui.red.primary.submit.button[disabled] {
            background-color: #d3d3d3 !important;
            color: #808080 !important;
        }

        .ui.red.primary.submit.button {
            background-color: #db2828 !important;
            color: #fff !important;
        }

        .ui.button {
            background-color: #db2828;
            color: white;
        }

        .ui.button:hover {
            background-color: #ff6961;
        }

        .ui.button.enabled {
            background-color: #ff6961;
        }
    </style>
</head>
<body>
    <div id="content">
        <div class="ui segment">
            <h2 class="ui header">Tabla de Datos</h2>
            <div class="ui buttons">
                <button id="btnExcel" class="ui button" disabled>Excel</button>
                <button id="btnGrafo" class="ui button" disabled>Grafo</button>
                <button id="btnTortuga" class="ui button" disabled>Tortuga</button>
</div>
```

```
        <div class="logo-egam">
          <div class="power-by">Powered by</div>
          <div class="capa-svg">
            <svg data-name="Grupo 1" xmlns="http://www.w3.org/2000/svg"
width="100" height="100" viewBox="0 0 703.377 256.281">
              <path id="e" class="cls-1" d="M751.833,559.46q8.057,0,11.678-
2.97t3.621-9.644a38.286,38.286,0,0,0-3.214-15.055,42.944,42.944,0,0,0-8.749-
13.265,45.851,45.851,0,0,0-15.625-10.62,49.661,49.661,0,0,0-18.88-3.54q-23.52,0-
37.191,12.9t-13.672,35.2q0,21.485,13.387,34.1t36.174,12.614q18.962,0,30.965-
6.307t12-15.829a9.238,9.238,0,0,0-3.133-7.2,11.564,11.564,0,0,0-8.016-2.807q-4.8,0-
11.8,4.476-2.118,1.385-3.256,2.034a33.98,33.98,0,0,1-6.714,2.808,23.607,23.607,0,0,1-
6.388.854q-8.871,0-13.265-4.354t-4.72-13.3Zm-42.44-29.744q4.027-4.353,10.946-
4.354,7.4,0,11.433,4.15t4.6,12.289H705.039Q705.363,534.07,709.393,529.716Z"
transform="translate(-669.813 -411.563)"></path>
              <path id="GAM" class="cls-2" d="M958.988,531.3q0-17.579-7.568-25.7t-
24.17-8.118h-23.8q-13.429,0-19.836,4.944t-
6.409,15.32q0,9.4,5.31,13.794t16.785,4.394h6.714a24.607,24.607,0,0,1-
10.926,15.931q-8.851,5.922-21.545,5.92-17.7,0-26.367-12.085t-8.667-36.5q0-
26.367,8.606-39.673t25.94-
13.306q11.961,0,23.315,10.254,1.708,1.465,2.686,2.32a10.518,10.518,0,0,0,.854.732q9
.888,8.668,19.9,8.667,11.472,0,18.188-6.286t6.714-16.907q0-18.677-19.775-31.067t-
50.9-12.39q-43.212,0-68.481,25.512t-
25.269,69.214q0,44.069,22.217,69.4t60.791,25.33q18.431,0,32.166-
5.066a61.211,61.211,0,0,0,23.5-
15.32q0.243,9.156,4.883,13.672t14.16,4.517q10.254,0,15.625-5.738t5.371-
16.6V531.3ZM1032.96,572.2h57.5l4.03,10.5a23.935,23.935,0,0,0,9.15,12.207q6.225,4.
149,15.38,4.151,11.715,0,18.86-6.714t7.14-17.822a38.5,38.5,0,0,0-.48-
6.226,23.55,23.55,0,0,0-1.35-5L1099,440.238q-5.61-15.867-11.84-21.179t-17.21-5.31h-
16.72q-10.995,0-17.52,5.737-6.54,5.739-11.78,20.752l-
43.7,123.047a19.427,19.427,0,0,0-1.1,4.639,49.508,49.508,0,0,0-
.366,6.347q0,11.354,7.019,18.067t18.977,6.714a26.174,26.174,0,0,0,15.02-
4.212,24.29,24.29,0,0,0,9.15-12.146Zm29.06-99.975,17.94,62.256h-
36.25Zm160.64,36.987q0-.243.12-5.005t0.12-9.155q0-6.836-.12-10.986t-0.36-
8.057q2.8,14.282,4.33,21.484t2.99,12.574l18.92,72.631q2.205,8.67,7.45,12.207t15.87,3
.54q10.62,0,16.17-3.723t7.75-12.268l18.8-74.463q1.47-5.37,3.3-13.305t4.15-18.677q-
0.12,10.011-.43,20.447t-
0.3,11.047V572.44q0,12.574,7.01,19.288t19.96,6.713q12.21,0,18.5-6.774t6.28-
19.959V439.018q0-12.205-6.77-18.616t-19.72-6.409h-21.24q-11.835,0-18.06,5.188t-
9.89,18.372l-19.53,68.725q-1.23,4.029-2.5,9.7-1.29,5.676-3.24,16.3-2.445-11.106-3.97-
17.7t-2.13-8.667l-18.92-67.139q-4.275-15.134-10.2-20.2t-18.37-5.066h-21.24q-
12.945,0-19.71,6.409-6.78,6.408-
6.78,18.615V571.708q0,13.183,6.29,19.959t18.49,6.774q12.945,0,19.96-6.713t7.02-
19.288V509.208Z" transform="translate(-669.813 -411.563)"></path>
              <circle id="Elipse_1" data-name="Elipse 1" class="cls-3" cx="40.187"
cy="203.437" r="8"></circle>
            </svg>
          </div>
        </div>
      </div>
```

```
        <form class="ui form" enctype="multipart/form-data"  style="left: 8px; top: -
1px">
            <input type="file" name="facturaPeritacion" id="file"
accept="application/vnd.ms-excel,application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"  />
            <div class="ui red primary submit button" id="btnUploadFile">Subir
Archivo</div>
            <div class="ui red primary submit button" id="btnDownloadJSON"
style="display: none">Descargar JSON</div>
            <div id="btnDownloadImage" class="ui red primary submit button"
style="display: none;">Descargar diagrama como imagen</div>
            <div id="excelContainer" runat="server" class="ui grid" style="margin-top:
20px; height: 686px;"></div>
            <div id="grafoContainer" style="width: 100%; height: 800px; display:
none;"></div>
            <div id="tortugaContainer" style="width: 100%; height: 800px; display:
none;"></div>
        </form>
    </div>
</body>
</html>
```

Default.aspx.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Web.UI;
using Newtonsoft.Json;
using NPOI.SS.UserModel;
using NPOI.XSSF.UserModel;
using System.Web;
using System.Text;
using System.Linq;

namespace ABDFinal
{
    public partial class WebForm1 : System.Web.UI.Page
    {

        protected void Page_Load(object sender, EventArgs e)
        {
            switch (Request["op"])
            {
                case "1":
                    EnviarFichero();
                    break;
            }
        }
```

```
        public Dictionary<string, string> idenToIndex = new Dictionary<string,
string>();

        private void EnviarFichero()
        {
            string Respuesta = string.Empty;
            HttpFileCollection files = Request.Files;
            string NombreTempFichero = string.Empty;
            HttpPostedFile file = files[0];
            var respuesta = new { Estado = "ok" };
            if (files.Count > 0)
            {
                try
                {
                    // Guardar el archivo en el servidor
                    string folderpath = Server.MapPath("~/Uploads/");
                    string filepath = folderpath + Path.GetFileName(files[0].FileName);
                    if (!Directory.Exists(folderpath))
                    {
                        // Si no existe, créalo
                        Directory.CreateDirectory(folderpath);
                    }
                    files[0].SaveAs(filepath);

                    // Diccionario para almacenar los valores, ahora vacío
                    Dictionary<string, List<string>> datosTabla = new Dictionary<string,
List<string>>();
                    Dictionary<string, string> entradas = new Dictionary<string, string>();
                    Dictionary<string, string> salidas = new Dictionary<string, string>();
                    Dictionary<string, List<string>> proceso = new Dictionary<string,
List<string>>();
                    Dictionary<string, List<string>> indicadores = new Dictionary<string,
List<string>>();
                    Dictionary<string, List<string>> usuarios = new Dictionary<string,
List<string>>();
                    Dictionary<string, List<string>> procedimiento = new
Dictionary<string, List<string>>();

                   LeerTablaExcel(filepath, datosTabla);

                    idenToIndex.Clear();

                    GrafoData grafo = BpmnBuilder.CrearGrafoDesdeExcel(datosTabla);
                    entradas = LeerEntradas(filepath, entradas);
                    salidas = LeerSalidas(filepath, salidas);
                    var resultadoProceso = LeerProceso(filepath, proceso);
                    string nombreProceso = resultadoProceso.Item1;
                    proceso = resultadoProceso.Item2;
                    indicadores = LeerIndicadores(filepath, indicadores);
                    usuarios = LeerUsuarios(filepath, usuarios);
                    procedimiento = LeerProcedimiento(filepath, procedimiento);
```

```
// Convertir los datos a formato JSON
                var jsonresult = new
                {
                    datosExcell = datosTabla,
                    Grafo = grafo,
                    Entradas = entradas,
                    Salidas = salidas,
                    Proceso = proceso,
                    Usuarios = usuarios,
                    Procedimiento = procedimiento,
                    nombreProceso = nombreProceso,
                    Estado = "ok"
                };

                string JsonRespuesta = string.Empty;

                JsonRespuesta = JsonConvert.SerializeObject(jsonresult,
Formatting.Indented);

                Response.ContentType = "application/json";
                // Response.AddHeader("Content-Disposition",
"attachment;filename=data.json");
                Response.Write(JsonRespuesta);
                try { Response.End(); }
                catch {}
            }
            catch (Exception ex)
            {
                var jsonresult = new
                {
                    error = ex.Message,
                    // Estado = "nok"
                };
            }
        }
    }

    public class GrafoData
    {
        public List<GoJsNode> Nodes { get; set; } = new List<GoJsNode>();
        public List<GoJsLink> Links { get; set; } = new List<GoJsLink>();
    }

    public class GoJsNode
    {
        public string key { get; set; }
        public string text { get; set; }
        public string category { get; set; }
        public List<string> predecessors { get; set; } = new List<string>();
    }
```

```
    public class GoJsLink
        {
            public string key { get; set; }
            public string from { get; set; }
            public string to { get; set; }
            public string text { get; set; }
        }

    public static class BpmnBuilder
        {
            public static GrafoData CrearGrafoDesdeExcel(Dictionary<string,
List<string>> datosTabla)
            {
                var grafo = new GrafoData();
                int totalFilas = datosTabla.First().Value.Count;

                var nodoInicio = new GoJsNode { key = "inicio", text = "Inicio", category
= "StartEvent" };
                var nodoFin = new GoJsNode { key = "fin", text = "Fin", category =
"EndEvent" };
                grafo.Nodes.Add(nodoInicio);
                grafo.Nodes.Add(nodoFin);

                for (int fila = 0; fila < totalFilas; fila++)
                {
                    var nodo = new GoJsNode();
                    string posteriores = null;
                    foreach (var kvp in datosTabla)
                    {
                        string nombreColumna = kvp.Key;
                        string valorCelda = kvp.Value[fila];

                                if (nombreColumna == "Num. Identificador") nodo.key =
                            valorCelda;
                        else if (nombreColumna == "Actividad / Tarea") nodo.text =
valorCelda;
                        else if (nombreColumna == "Tipo") nodo.category = valorCelda;

                        else if (nombreColumna == "Act. Posterior" &&
                !string.IsNullOrEmpty(valorCelda))
                        {
                            posteriores = valorCelda;
                        }
                        else if (nombreColumna == "Act. Posterior" && valorCelda == "fin")
                        {
                            grafo.Links.Add(new GoJsLink { key = "fin" + nodo.key, from =
nodo.key, to = "fin", text = "" });
                        }
                    }
```

```
        if (!string.IsNullOrEmpty(posteriores))
                {
                    // Verificar si hay un nodo previo
                    if (fila > 0)
                    {
                        var nodoPrevio = grafo.Nodes[grafo.Nodes.Count - 1];
                        var posterioresPrevios = datosTabla["Act. Posterior"][fila - 1];

                        // Si el nodo actual y el previo tienen los mismos posteriores, crea
un nodo intermedio
                        if (posteriores == posterioresPrevios)
                        {
                            var nodoIntermedio = new GoJsNode
                            {
                                key = nodo.key + "_intermedio",
                                text = "",
                                category = "IntermediateEvent"
                            };
                            grafo.Nodes.Add(nodoIntermedio);

                            // Conectar nodos originales al nodo intermedio
                            grafo.Links.Add(new GoJsLink { key = "link_" + nodoPrevio.key
+ "_" + nodoIntermedio.key, from = nodoPrevio.key, to = nodoIntermedio.key, text =
"" });
                            grafo.Links.Add(new GoJsLink { key = "link_" + nodo.key + "_" +
nodoIntermedio.key, from = nodo.key, to = nodoIntermedio.key, text = "" });

                            // Eliminar conexiones originales de los nodos a los posteriores
                            grafo.Links.RemoveAll(link => link.from == nodoPrevio.key &&
posteriores.Split(';').Contains(link.to));
                            grafo.Links.RemoveAll(link => link.from == nodo.key &&
posteriores.Split(';').Contains(link.to));

                            // Conectar el nodo intermedio a los posteriores
                            foreach (var successor in posteriores.Split(';'))
                            {
                                grafo.Links.Add(new GoJsLink { key = "link_" +
nodoIntermedio.key + "_" + successor, from = nodoIntermedio.key, to = successor,
text = "" });
                            }

                            grafo.Nodes.Add(nodo); // Agregar el nodo actual al grafo

                            continue; // Evitar agregar enlaces adicionales para este nodo
en esta iteración
                        }
                    }
```

```
                    // Agregar enlaces regulares para nodos no intermedios
                    foreach (var successor in posteriores.Split(';'))
                    {
                        grafo.Links.Add(new GoJsLink { key = "link_" + nodo.key + "_" +
successor, from = nodo.key, to = successor, text = "" });
                    }
                }

                grafo.Nodes.Add(nodo);
                AddLinks(grafo, nodo, fila);
            }

            grafo.Links.Add(new GoJsLink { key = "enlace_inicio", from =
nodoInicio.key, to = grafo.Nodes[2].key, text = "" });
            return grafo;
        }
    }

    private static void AddLinks(GrafoData grafo, GoJsNode nodo, int fila)
    {
        if (fila > 0)
        {
            foreach (string predecessor in nodo.predecessors)
            {
                var enlace = new GoJsLink { key = "enlace_" + predecessor +
grafo.Nodes.Last().key, from = predecessor, to = grafo.Nodes.Last().key, text = "" };
                grafo.Links.Add(enlace);
            }
        }
    }

private static void LeerTablaExcel(string filePath, Dictionary<string, List<string>>
datos)
    {
        // Cargar el archivo Excel
        using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
        {
            IWorkbook workbook = new XSSFWorkbook(fileStream);

            // Obtener la hoja de trabajo (worksheet)
            ISheet worksheet = workbook.GetSheetAt(0); // Queremos la primera
hoja de excel

            // Obtener el número total de columnas y filas
            int totalFilas = worksheet.PhysicalNumberOfRows;
            int totalColumnas = worksheet.GetRow(2).PhysicalNumberOfCells;
```

```
            // Iterar sobre las columnas de la tercera fila (nombres de las columnas)
            for (int col = 1; col < totalColumnas; col++)
            {
                string nombreColumna = worksheet.GetRow(2).GetCell(col).StringCellValue;
                if (nombreColumna == "Num. Identificador" || nombreColumna == "Actividad / Tarea" || nombreColumna == "Tipo" || nombreColumna == "Responsable" || nombreColumna == "Act. Predecesora" || nombreColumna == "Act. Posterior" || nombreColumna == "Opción SI" || nombreColumna == "Opción NO")
                {
                    datos[nombreColumna] = new List<string>();
                }
            }
            // Iterar sobre las filas (empezando desde la cuarta fila)
            for (int fila = 3; fila <= totalFilas; fila++)
            {
                // Iterar sobre las columnas empezando desde la segunda columna
                for (int col = 1; col <= totalColumnas; col++)
                {
                    string nombreColumna = worksheet.GetRow(2)?.GetCell(col)?.StringCellValue;
                    string ValorCelda = worksheet.GetRow(fila)?.GetCell(col)?.ToString();

                    if (nombreColumna == "Num. Identificador" || nombreColumna == "Actividad / Tarea" || nombreColumna == "Tipo" || nombreColumna == "Responsable" || nombreColumna == "Act. Predecesora" || nombreColumna == "Act. Posterior" || nombreColumna == "Opción SI" || nombreColumna == "Opción NO")
                    {
                        if (ValorCelda != "" && ValorCelda != null)
                        {
                            // Agregar el valor al diccionario bajo el nombre de la columna correspondiente
                            datos[nombreColumna].Add(ValorCelda);
                        }
                        else if (nombreColumna == "Actividad / Tarea" && ValorCelda == "")
                        {
                            datos["Num. Identificador"].Remove(worksheet.GetRow(fila).GetCell(col - 1).ToString());
                            fila++;
                            col = 0;
                        }
                        else if (ValorCelda == null)
                        {
                            //No hace nada
                        }
                        else
                        {
                            datos[nombreColumna].Add(ValorCelda);
                        }
                    }
                }
```

```
        private static Dictionary<string, string> LeerEntradas(string filePath,
Dictionary<string, string> entradas)
        {
            using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
            {
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(3); // Queremos la cuarta hoja de
excel
                int totalFilas = worksheet.PhysicalNumberOfRows;
                int col = 1;
                for (int fil = 4; fil <= totalFilas; fil++)
                {
                    string Fuente = worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    string Entrada = worksheet.GetRow(fil).GetCell(col + 1).StringCellValue;
                    if (Fuente != "" && Entrada != "")
                    {
                        entradas.Add(Fuente, Entrada);
                    }
                }

            }
            return entradas;
        }
        private static Dictionary<string, string> LeerSalidas(string filePath,
Dictionary<string, string> salidas)
        {
            using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
            {
                IWorkbook workbook = new XSSFWorkbook(fileStream);
                ISheet worksheet = workbook.GetSheetAt(3);
                int totalFilas = worksheet.PhysicalNumberOfRows;
                int col = 4;
                for (int fil = 4; fil <= totalFilas; fil++)
                {
                    string Salidas = worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    string Receptores = worksheet.GetRow(fil).GetCell(col +
1).StringCellValue;
                    if (Salidas != "" && Receptores != "")
                    {
                        salidas.Add(Salidas, Receptores);
                    }
                }
            }
            return salidas;
        }
```

```
        private static Tuple<string, Dictionary<string, List<string>>> LeerProceso(string
filePath, Dictionary<string, List<string>> proceso)
        {
            string nombreProceso = string.Empty;

            using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
            {
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(2);
                int totalFilas = worksheet.PhysicalNumberOfRows;
                int col = 1;
                string dato;
                string dato2;

                // Leer el nombre del proceso de la celda "C3"
                nombreProceso = worksheet.GetRow(2).GetCell(2).StringCellValue;

                for (int fil = 2; fil < 5; fil++)
                {
                    dato = worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    dato2 = worksheet.GetRow(fil).GetCell(2).StringCellValue;
                    List<string> detalles = new List<string> { dato2 };  // Nueva instancia de
List<string>
                    proceso.Add(dato, detalles);
                }

                dato = worksheet.GetRow(6).GetCell(1).StringCellValue;
                dato2 = worksheet.GetRow(7).GetCell(1).StringCellValue;
                proceso.Add(dato, new List<string> { dato2 });  // Nueva instancia de
List<string>

                dato = worksheet.GetRow(10).GetCell(1).StringCellValue;
                List<string> ultimosDetalles = new List<string>();  // Nueva instancia de
List<string>
                for (int fil = 11; fil < totalFilas; fil++)
                {
                    try
                    {
                        dato2 = worksheet.GetRow(fil).GetCell(1).StringCellValue;
                    }
                    catch (Exception ex)
                    {
                        continue;
                    }
                    if (dato2 != "")
                    {
                        ultimosDetalles.Add(dato2);
                    }
                }
                proceso.Add(dato, ultimosDetalles);
```

```
            return Tuple.Create(nombreProceso, proceso);
        }
    }
    private static Dictionary<string, List<string>> LeerIndicadores(string filePath,
Dictionary<string, List<string>> indicadores)
    {
        using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
        {
            IWorkbook workbook = new XSSFWorkbook(fileStream);

            // Obtener la hoja de trabajo (worksheet)
            ISheet worksheet = workbook.GetSheetAt(4); // Queremos la cuarta hoja
de excel
            int totalFilas = worksheet.PhysicalNumberOfRows;

            for (int col = 1; col < 6; col++)
            {
                string nombreColumna =
worksheet.GetRow(1).GetCell(col).StringCellValue;
                List<string> detalles = new List<string>(); // Nueva instancia de
List<string> para cada columna

                for (int fil = 2; fil < totalFilas; fil++)
                {
                    var cell = worksheet.GetRow(fil).GetCell(col);
                    string dato;

                    if (cell == null)
                    {
                    }
                    else if (cell.CellType == CellType.Numeric)
                    {
                        dato = cell.NumericCellValue.ToString();
                        detalles.Add(dato);
                    }
                    else if (cell.StringCellValue == "" || cell.StringCellValue == "Nota: Se
puede definir un máximo de 4 indicadores por año, uno de cada tipo (U.medida),
para cada proceso.")
                    {
                    }
                    else
                    {
                        dato = cell.StringCellValue;
                        detalles.Add(dato);
                    }
                }
```

```
                indicadores.Add(nombreColumna, detalles);
            }
        }
        return indicadores;
    }
    private static Dictionary<string, List<string>> LeerUsuarios(string filePath,
Dictionary<string, List<string>> usuarios)
    {
        using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
        {
            IWorkbook workbook = new XSSFWorkbook(fileStream);

            // Obtener la hoja de trabajo (worksheet)
            ISheet worksheet = workbook.GetSheetAt(5);
            int totalFilas = worksheet.PhysicalNumberOfRows;
            for (int col = 1; col < 4; col++)
            {
                string nombreColumna =
worksheet.GetRow(1).GetCell(col).StringCellValue;
                List<string> detalles = new List<string>();
                for (int fil = 2; fil <= totalFilas; fil++)
                {
                    string cell = worksheet.GetRow(fil).GetCell(col).StringCellValue;
                    string dato;

                    if (cell == "")
                    {
                        // No hacer nada si la celda está vacía
                    }
                    else
                    {
                        // Si no es numérico, obtiene el valor como string directamente
                        dato = cell;
                        detalles.Add(dato);
                    }
                }
                usuarios.Add(nombreColumna, detalles);
            }
            return usuarios;
        }
    }
```

```
private static Dictionary<string, List<string>> LeerProcedimiento(string filePath,
Dictionary<string, List<string>> datos)
        {
            // Cargar el archivo Excel
            using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
            {
                IWorkbook workbook = new XSSFWorkbook(fileStream);

                // Obtener la hoja de trabajo (worksheet)
                ISheet worksheet = workbook.GetSheetAt(0); // Queremos la primera
hoja de excel

                // Obtener el número total de columnas y filas
                int totalFilas = worksheet.PhysicalNumberOfRows;
                int totalColumnas = worksheet.GetRow(2).PhysicalNumberOfCells;

                // Iterar sobre las columnas de la tercera fila (nombres de las
columnas)
                for (int col = 1; col <= totalColumnas; col++)
                {
                    string nombreColumna =
worksheet.GetRow(2).GetCell(col).StringCellValue;
                    datos[nombreColumna] = new List<string>();
                }

                // Iterar sobre las filas (empezando desde la cuarta fila)
                for (int fila = 3; fila <= totalFilas; fila++)
                {
                    // Iterar sobre las columnas empezando desde la segunda columna
                    for (int col = 1; col <= totalColumnas; col++)
                    {
                        string nombreColumna =
worksheet.GetRow(2)?.GetCell(col)?.StringCellValue;
                        string ValorCelda =
worksheet.GetRow(fila)?.GetCell(col)?.ToString();

                        if (ValorCelda != "" && ValorCelda != null)
                        {
                            // Agregar el valor al diccionario bajo el nombre de la columna
correspondiente
                            datos[nombreColumna].Add(ValorCelda);
                        }
                return datos;
            }
        }
    }
}
```

```
                else if (nombreColumna == "Actividad / Tarea" && ValorCelda
== "")
                {
                    datos["Num.
Identificador"].Remove(worksheet.GetRow(fila).GetCell(col - 1).ToString());
                    fila++;
                    col = 0;
                }
                else if (ValorCelda == null)
                {
                    //No hace nada
                }
                else
                {
                    datos[nombreColumna].Add(ValorCelda);
                }
            }
        }

        return datos;
    }
  }
}
```

Funciones.js:

```
var datosObtenidos = null;
var flujoDiagrama = null;
var tortugaDiagrama = null;

$(function () {
    try {
        initializeForm();
        initializeButtons();
    } catch (e) {
        console.error(e.message);
    }
});

function initializeForm() {
    $('.ui.form').form();
    $('.ui.form').on('submit',  handleOnSubmit);
}
```

```
function initializeButtons() {
    $('#btnExcel').on('click', function () {
        showContainer('excel');
    }).hover(function () {
        $(this).toggleClass('hovered');
    });

    $('#btnGrafo').on('click', function () {
        showContainer('grafo');
    }).hover(function () {
        $(this).toggleClass('hovered');
    });

    $('#btnTortuga').on('click', function () {
        showContainer('tortuga');
    }).hover(function () {
        $(this).toggleClass('hovered');
    });
    $('#btnUploadFile').on('click', function () {
        uploadFile();
    });
    $('#btnDownloadJSON').on('click', function () {
        downloadJSON();
    });
    $('#btnDownloadImage').on('click', function () {
        downloadImage();
    });
}

function showContainer(type) {
    $('#excelContainer, #grafoContainer, #tortugaContainer').hide();

    console.log("Switching to view:", type);

    if (type === 'excel') {
        $('#excelContainer').show();
    } else if (type === 'grafo') {
        $('#grafoContainer').show();
        if (datosObtenidos && datosObtenidos.Grafo) {
            generarDiagramaFlujo(datosObtenidos.Grafo);
        } else {
            console.error('Invalid or missing graph data.');
        }
```

```
    } else if (type === 'tortuga') {
        $('#tortugaContainer').show();
        if (datosObtenidos) {
            generarDiagramaTortuga(datosObtenidos);
        } else {
            console.error('Invalid or missing graph data.');
        }
    }
    if (type === 'grafo' || type === 'tortuga') {
        $('#btnDownloadImage').show();
    } else {
        $('#btnDownloadImage').hide();
    }
}

function MostrarDatos(data, vistaTabla) {
    showTable(data.datosExcell);
}

function showTable(datosExcell) {
    let html = "<table class='ui striped table'><thead><tr>";

    for (let key in datosExcell) {
        html += `<th>${key}</th>`;
    }
    html += "</tr></thead><tbody>";

    for (let i = 0; i < datosExcell[Object.keys(datosExcell)[0]].length; i++) {
        html += "<tr>";
        for (let key in datosExcell) {
            html += `<td>${datosExcell[key][i]}</td>`;
        }
        html += "</tr>";
    }

    html += "</tbody></table>";

    $("#excelContainer").html(html);
    showContainer('excel');
}
```

```
function handleOnSubmit(e) {
    e.preventDefault();
    if ($('.ui.form').form('is  valid')) {
        let form = document.querySelector('.form');
        let Datos = new FormData(form);
        Datos.append("op", "1");

        $.ajax({
            type: "POST",
            url: "Default.aspx",
            processData: false,
            contentType: false,
            data: Datos,
            success: handleSuccess,
            error: handleError
        });
    } else {
        console.error('Form is incomplete');
    }
}
function handleSuccess(data, txtStatus, jqXHR) {
    if (data.Estado === "ok") {
        datosObtenidos = data;
        MostrarDatos(data, true);
        enableButtons();
        if ($('#grafoContainer').is(':visible')) {
            if (datosObtenidos && datosObtenidos.Grafo) {
                generarDiagramaFlujo(datosObtenidos.Grafo);
            } else {
                console.error('Invalid or missing graph data.');
            }
        }

        if ($('#tortugaContainer').is(':visible')) {
            if (datosObtenidos) {
                generarDiagramaTortuga(datosObtenidos);
            } else {
                console.error('Invalid or missing graph data.');
            }
        }
    } else if (data.Estado === "nok") {
        console.log(data.error);
    }
}
```

```
function handleError(jqXHR, textStatus, errorThrown) {
    console.log(errorThrown);
}

function downloadJSON() {
    if (datosObtenidos) {
        var blob = new Blob([JSON.stringify(datosObtenidos)], { type: 'application/json'
});
        var link = document.createElement('a');
        link.href = URL.createObjectURL(blob);
        link.download = 'datos.json';
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    } else {
        console.error('No hay datos disponibles para descargar');
    }
}
function downloadImage() {
    let container = $('#grafoContainer').is(':visible') ? $('#grafoContainer') :
$('#tortugaContainer');

    if (!container.is(':visible')) {
        console.error('No hay diagrama visible para descargar');
        return;
    }

    html2canvas(container[0]).then(canvas => {
        let link = document.createElement('a');
        link.download = 'diagrama.png';
        link.href = canvas.toDataURL();
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    });
}


function uploadFile() {
    let input = $('#file')[0];
    let file = input.files[0];

    if (!file) {
        alert('Seleccione un archivo primero');
        return;
    }
```

```
let formData = new FormData();
    formData.append('facturaPeritacion', file);
    formData.append('op', '2');

    $.ajax({
        type: 'POST',
        url: 'Default.aspx',
        data: formData,
        contentType: false,
        processData: false,
        success: handleSuccess,
        error: handleError
    });
}


function enableButtons() {
    $('#btnDownloadJSON').show();
    $('#btnExcel').prop('disabled', false);
    $('#btnGrafo').prop('disabled', false);
    $('#btnTortuga').prop('disabled', false);
    }


function generarDiagramaFlujo(GrafoData) {
    if (flujoDiagrama) {
        flujoDiagrama.div = null; // Destruir el diagrama existente
        flujoDiagrama = null;
    }
    if (!GrafoData || !GrafoData.Nodes || !GrafoData.Links) {
        console.error('Datos de grafo no válidos o faltantes.');
        return;
    }

    var $ = go.GraphObject.make;
    flujoDiagrama = $(go.Diagram, "grafoContainer", {
        "undoManager.isEnabled": true,
        layout: $(go.LayeredDigraphLayout, {
            direction: 0,
            layeringOption: go.LayeredDigraphLayout.LayerLongestPathSource,
            columnSpacing: 30,
            isRouting: true,
            packOption: go.LayeredDigraphLayout.PackMedian,
        })
    });
```

```
function makePort(name, spot, output, input) {
        return $(go.Shape, "Circle",
            {
                fill: "transparent",
                stroke: null,
                desiredSize: new go.Size(8, 8),
                alignment: spot, alignmentFocus: spot,
                portId: name,
                fromSpot: spot, toSpot: spot,
                fromLinkable: output, toLinkable: input,
                cursor: "pointer"
            });
    }

    function createNodeTemplate(shapeType, fillColor) {
        return $(go.Node, "Spot",
            $(go.Panel, "Auto",
                $(go.Shape, shapeType, { fill: fillColor, stroke: "black" }),
                $(go.TextBlock, { margin: 5, editable: true, textAlign: "center", wrap:
go.TextBlock.WrapFit, width: 100 },
                    new go.Binding("text", "text", function (t) {
                        var words = t.split(" ");
                        if (words.length > 2) {
                            return words.slice(0, 2).join(" ") + "\n" + words.slice(2).join(" ");
                        } else {
                            return t;
                        }
                    })
                )
            ),
            makePort("T", go.Spot.Top, true, true),
            makePort("L", go.Spot.Left, true, true),
            makePort("R", go.Spot.Right, true, true),
            makePort("B", go.Spot.Bottom, true, true)
        );
    }

flujoDiagrama.nodeTemplateMap.add("StartEvent", createNodeTemplate("Circle",
"green"));
    flujoDiagrama.nodeTemplateMap.add("EndEvent", createNodeTemplate("Circle",
"red"));
    flujoDiagrama.nodeTemplateMap.add("Estandar", createNodeTemplate("Rectangle",
"white"));
    flujoDiagrama.nodeTemplateMap.add("Decisión", createNodeTemplate("Diamond",
"yellow"));
    flujoDiagrama.nodeTemplateMap.add("Aprobación",
createNodeTemplate("Diamond", "cyan"));
    flujoDiagrama.nodeTemplateMap.add("IntermediateEvent",
createNodeTemplate("Circle", "purple"));
```

```
flujoDiagrama.linkTemplate =
        $(go.Link,
            {
                routing: go.Link.AvoidsNodes, corner: 5,
                relinkableFrom: true,
                relinkableTo: true,
                curviness: 20,
                adjusting: go.Link.End
            },
            $(go.Shape),
            $(go.Shape, { toArrow: "Standard" }),
            $(go.TextBlock, { margin: 3, editable: true }, new go.Binding("text", "text"))
        );

    var model = $(go.GraphLinksModel);
    model.nodeDataArray = GrafoData.Nodes;
    model.linkDataArray = GrafoData.Links;

    flujoDiagrama.model = model;
}

function generarDiagramaTortuga(datos) {
    if (tortugaDiagrama) {
        tortugaDiagrama.div = null;
        tortugaDiagrama = null;
    }
    if (!datos) {
        console.error('Datos de grafo no válidos o faltantes.');
        return;
    }
    var $ = go.GraphObject.make;

    tortugaDiagrama = $(go.Diagram, 'tortugaContainer', {
        layout: $(DoubleTreeLayout, {
            vertical: true,
            directionFunction: (node) => node.data.dir === 'down',
        }),
    });

    const graygrad = $(go.Brush, 'Linear', { 0: '#F5F5F5', 1: '#F1F1F1' });
    const bluegrad = $(go.Brush, 'Linear', { 0: '#CDDAF0', 1: '#91ADDD' });
    const yellowgrad = $(go.Brush, 'Linear', { 0: '#FEC901', 1: '#FEA200' });
    const whitegrad = $(go.Brush, 'Linear', { 0: '#FFFFFF', 1: '#FFFFFF' });
    const greengrad = $(go.Brush, 'Linear', { 0: '#B0E57C', 1: '#86C55A' });
    const redgrad = $(go.Brush, 'Linear', { 0: '#FF7F7F', 1: '#FF4C4C' });
    const orangegrad = $(go.Brush, 'Linear', { 0: '#FFD580', 1: '#FFA500' });
    const purplegrad = $(go.Brush, 'Linear', { 0: '#D4A5E8', 1: '#B87BCB' });
```

```
    tortugaDiagrama.nodeTemplate = $(go.Node,
        'Auto',
        { isShadowed: true },
        $(go.Shape,
            'RoundedRectangle',
            { fill: graygrad, stroke: '#D8D8D8', width: 150, height: 50 },
            new go.Binding('fill', 'color')
        ),
        $(go.TextBlock, { margin: 10, font: 'bold 14px Helvetica, bold Arial, sans-serif' },
new go.Binding('text', 'key'))
    );

    tortugaDiagrama.linkTemplate = $(go.Link,
        { selectable: false },
        $(go.Shape, { strokeWidth: 2 },
            new go.Binding('stroke', '', (link) => {
                // Obtener el color del nodo al que apunta el enlace
                const toNode = link.toNode;
                if (toNode) {
                    if (toNode.data.key === 'Proceso') {
                        // Si el nodo de destino es 'Proceso', usar el color de su padre
                        const parentNode = toNode.findTreeParentNode();
                        return parentNode.data.color;
                    } else {
                        // En otros casos, usar el color del nodo de destino
                        return toNode.data.color;
                    }
                }
                return 'black'; // Color por defecto si no se encuentra el nodo de destino
            })
        )
    );

    const nodes = [
        { key: 'Proceso', color: whitegrad },
        { key: 'Fuentes', parent: 'Proceso', dir: 'up', color: bluegrad },
        { key: 'Entradas', parent: 'Proceso', dir: 'up', color: greengrad },
        { key: 'Recursos', parent: 'Proceso', dir: 'up', color: purplegrad },
        { key: 'Usuarios', parent: 'Proceso', dir: 'down', color: yellowgrad },
        { key: 'Salidas', parent: 'Proceso', dir: 'down', color: redgrad },
        { key: 'Receptores', parent: 'Proceso', dir: 'down', color: orangegrad }
    ];

  if (datos.Entradas) {
        for (const [key, value] of Object.entries(datos.Entradas)) {
            nodes.push({ key: value, parent: 'Entradas', dir: 'up', color: greengrad });
            nodes.push({ key: key, parent: 'Fuentes', dir: 'up', color: bluegrad });
        }
    }
```

```
  for (const recurso of datos.Proceso.Recursos) {
      nodes.push({ key: recurso, parent: 'Recursos', dir: 'up', color: purplegrad });
  }

  if (datos.Salidas) {
      for (const [key, value] of Object.entries(datos.Salidas)) {
          nodes.push({ key: value, parent: 'Receptores', dir: 'down', color: orangegrad });
          nodes.push({ key: key, parent: 'Salidas', dir: 'down', color: redgrad });
      }
  }

  if (datos.Usuarios && Array.isArray(datos.Usuarios.Nombre)) {
      for (const usuario of datos.Usuarios.Nombre) {
          nodes.push({ key: usuario, parent: 'Usuarios', dir: 'down', color: yellowgrad });
      }
  }

  tortugaDiagrama.model = new go.TreeModel(nodes);
}
```