HOJA DE CÁLCULO

1a ENTREGA

PROP

Subgrupo 1.4
Autores:
Gallardo Peña, Daniel (daniel.gallardo.pena@estudiantat.upc.edu)
Pérez Barroso, David (david.perez.barroso@estudiantat.upc.edu)
Risueño Martín, Iván (ivan.risueno@estudiantat.upc.edu)
Torra Garcia, Joaquim (joaquim.torra@estudiantat.upc.edu)

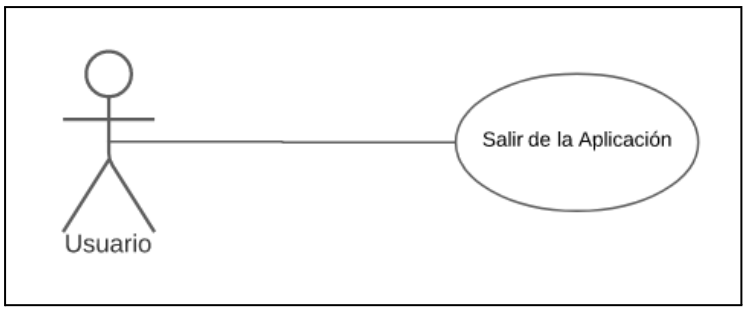
1a Entrega - 24 de Abril del 2022
Versión del entregable: 1.1

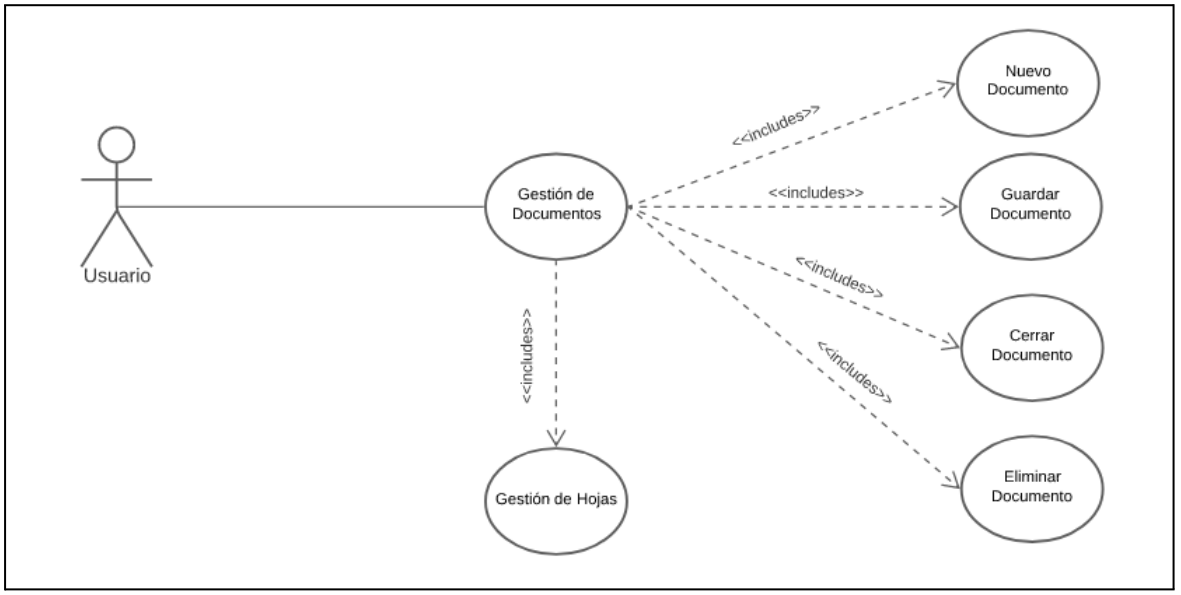
Índice

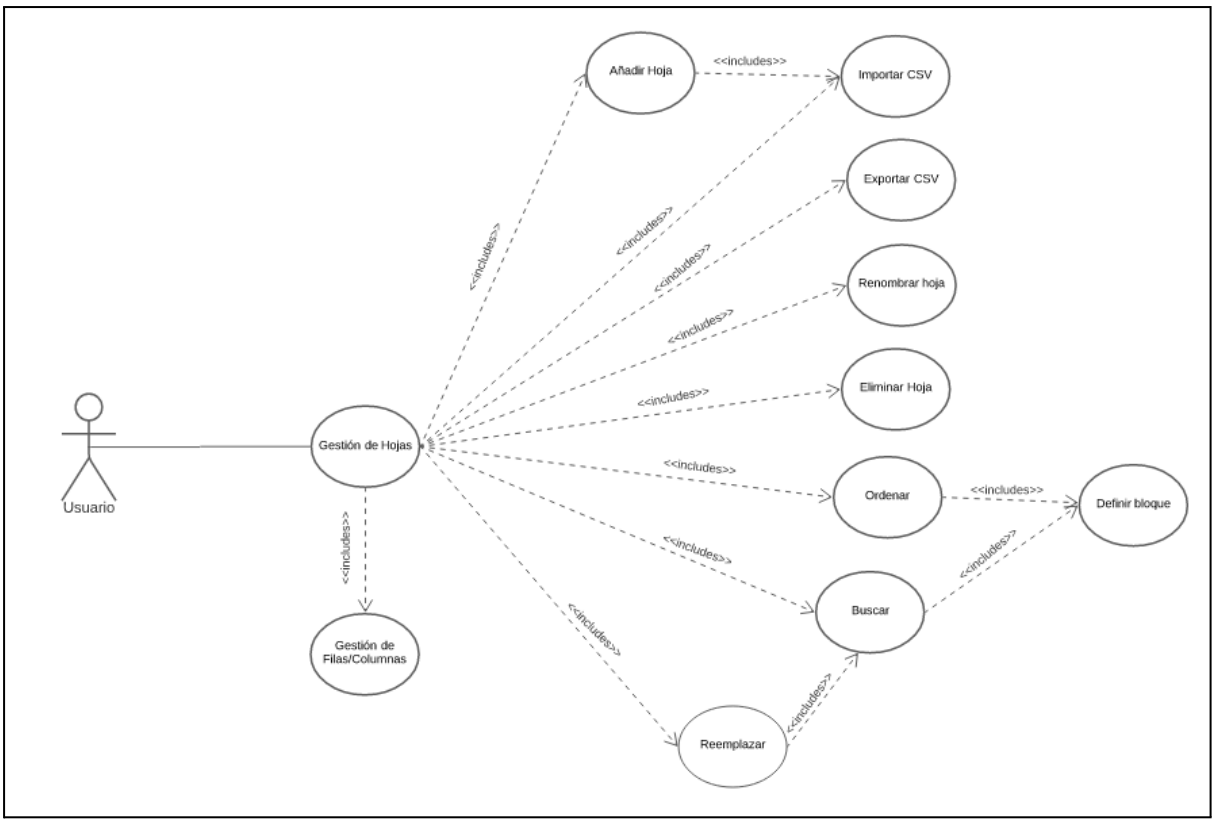
1. Diagrama de casos de uso

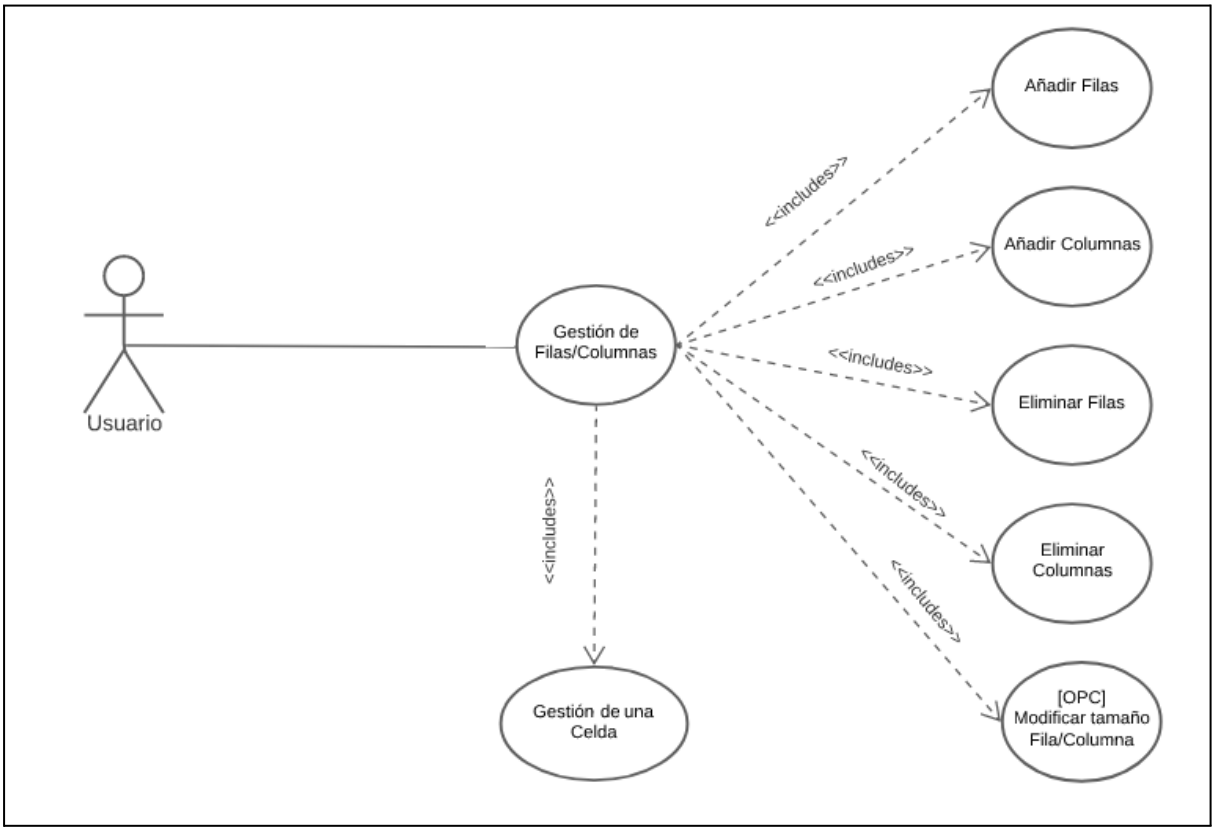
1.1 Actores

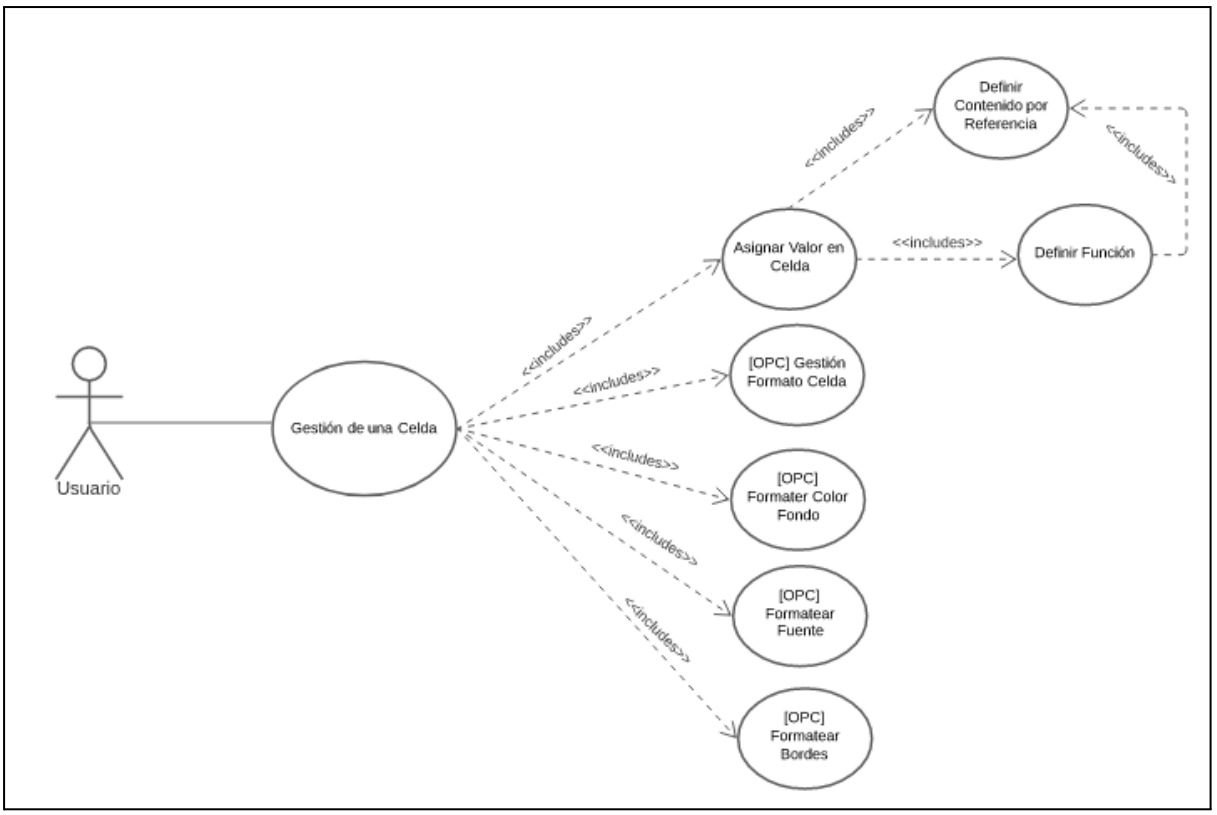
1.2 Diagrama de casos de uso



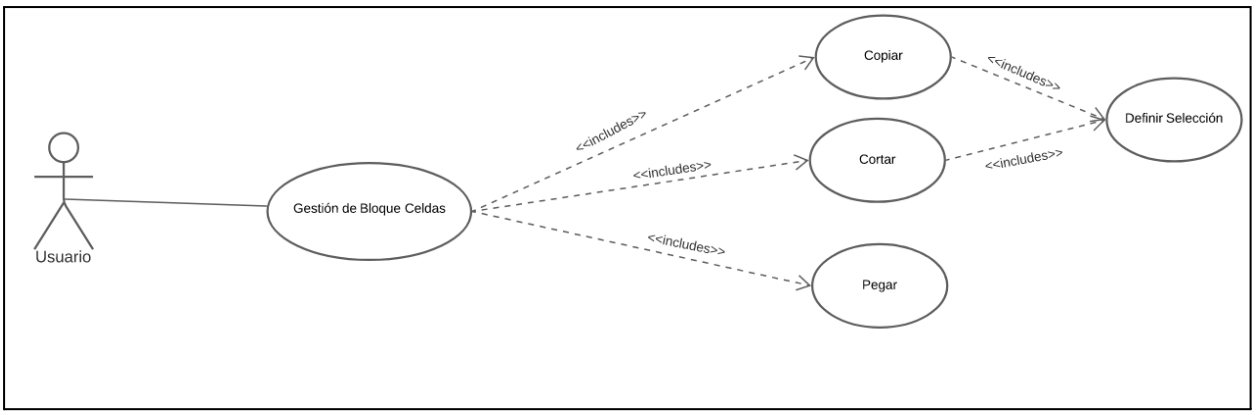



Usuario
Gestión de Hojas
Añadir Hoja
Importar CSV
Exportar CSV
Renombrar hoja
Eliminar Hoja
Ordenar
Definir bloque
Buscar
Reemplazar
Gestión de Filas/Columnas
<<includes>>


Usuario
Gestión de Filas/Columnas
Añadir Filas
Añadir Columnas
Eliminar Filas
Eliminar Columnas
[OPC] Modificar tamaño Fila/Columna
Gestión de una Celda
<<includes>>


Usuario
Gestión de una Celda
Definir Contenido por Referencia
Asignar Valor en Celda
Definir Función
[OPC] Gestión Formato Celda
[OPC] Formater Color Fondo
[OPC] Formatear Fuente
[OPC] Formatear Bordes
<<includes>>


Definir Función
Definir (Unitaria) F. Truncamiento
Definir (Unitaria) F. Conversión
Definir (Unitaria) F. Extracción fechas
Definir (Unitaria) F. Días de la semana
Definir (Unitaria) F. Reemplazamientos
Definir (Unitaria) F. Longitud
Definir (N-aria) F. estadísticas
[OPC]Definir (N-aria) F. aritméticas
[OPC]Definir (N-aria) F. comparación
<<includes>>



1.3 Descripción de casos de uso

Nombre	**Salir de la Aplicación**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción cerrar aplicación. 2. El sistema cierra la ventana gráfica y para la ejecución de la aplicación.
Extensiones	2a1. Si hay cambios no guardados, el sistema avisa al usuario y pregunta qué hacer: a. Cerrar sin guardar b. Guardar cambios y cerrar c. Cancelar acción

Nombre	**Nuevo Documento**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de crear un nuevo documento. 2. El sistema abre un menú emergente con las siguientes opciones: a. Crear documento en blanco. b. Cargar un documento a partir de un fichero. 3. El usuario selecciona crear un nuevo documento en blanco. 4. El sistema le pregunta al usuario cuantas filas y columnas desea, con valores por defecto 50x50. 5. El usuario introduce, si le es conveniente, el número de filas y el número de columnas. 6. El sistema crea un documento con una hoja en blanco que dispone de tantas filas y columnas como valores en los campos.
Extensiones	1a1. Si hay un documento abierto con cambios no guardados, el sistema advertirá al usuario de que estos se perderán y pedirá una confirmación. 1a2. Si el usuario decide no continuar, vuelve a donde estaba. 3a1. El usuario selecciona cargar un documento.

	3a2. Se abre un menú con los directorios del PC del usuario. 3a3. El usuario selecciona el fichero que desea cargar como documento. 3a4. El sistema carga el documento con el contenido del fichero seleccionado. Este tiene que ser del tipo XML. 3a4a Si el usuario indica un fichero no apto, muestra un mensaje de error y vuelve al paso 3a2. 5a1. Si el usuario introduce valores no numéricos o negativos en los campos de filas y columnas, el sistema devuelve un error y vuelve al paso 4.

Nombre	**Guardar Documento**
Actor	Usuario
Precondición	El documento tiene que haberse modificado desde la última vez que se guardó.
Comportamiento	1. El usuario informa que quiere guardar el documento. 2. Si el documento era nuevo y no había sido guardado aún, el sistema abre un menú con los directorios del PC del usuario. 3. El usuario navega hasta el directorio donde desea guardar el documento y le da un nombre. 4. El sistema guarda en disco el documento en formato XML en el path seleccionado y le asigna el nombre indicado.

Errores posibles y cursos alternativos	2a. Si el fichero ya había sido guardado en algún fichero previamente o había sido cargado de algún fichero, el sistema lo sobreescribe directamente. 4a. Si ya existe un archivo con el mismo nombre en el directorio indicado, el sistema pedirá una confirmación para sobrescribirlo.

Nombre	**Cerrar Documento**
Actor	Usuario
Precondición	Tiene que haber un documento abierto.
Comportamiento	2. El usuario selecciona la opción de cerrar el documento. 3. Si hay cambios no guardados, el sistema avisa al usuario y pregunta qué hacer: a. Cerrar sin guardar b. Guardar cambios y cerrar 4. El usuario indica que quiere hacer y el sistema actúa correspondientemente. 5. El sistema vuelve al menú de inicio.
Errores posibles y cursos alternativos	3a. Si el usuario cambia de opinión, puede cancelar la acción y vuelve al documento como estaba.

Nombre	**Eliminar Documento**
Actor	Usuario
Precondición	Tiene que haber un documento abierto.

Comportamiento	1. El usuario selecciona la opción de eliminar el documento. 2. El sistema pedirá una confirmación, avisando de que esta opción no es reversible. 3. El usuario confirma la acción. 4. El sistema borra el documento que está abierto actualmente y vuelve al menú de inicio.
Errores posibles y cursos alternativos	2a. Si el usuario decide no borrar el documento y vuelve a donde estaba. 4a. Si el documento se ha guardado o se ha cargado desde un fichero, también se elimina dicho fichero de la memoria del PC del usuario.

Nombre	**Añadir Hoja**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de añadir una nueva hoja. 2. El sistema abre un menú emergente con las siguientes opciones: a. Crear hoja en blanco. b. Cargar una hoja a partir de un fichero. 3. El usuario selecciona crear una nueva hoja en blanco. 4. El sistema le pregunta al usuario cuantas filas y columnas desea, con valores por defecto 50x50. 5. El usuario introduce, si le es conveniente, el número de filas y el número de columnas.

	6. El sistema crea una nueva hoja en blanco el documento existente o que dispone de tantas filas y columnas como valores en los campos.
Errores posibles y cursos alternativos	3a1. El usuario selecciona cargar una hoja. 3a2. Se abre un menú con los directorios del PC del usuario. 3a3. El usuario selecciona el fichero que desea cargar como hoja. 3a4. El sistema carga el documento con el contenido del fichero seleccionado. Este tiene que ser del tipo CSV o XML. 3a4a Si el usuario indica un fichero no apto, muestra un mensaje de error y vuelve al paso 3a2. 5a1. Si el usuario introduce valores erróneos en los campos de filas y columnas, el sistema devuelve un error.

Nombre	**Quitar Hoja**
Actor	Usuario
Precondición	El documento tiene almenos una hoja.
Comportamiento	1. El usuario selecciona una hoja para eliminar. 2. El sistema pedirá una confirmación, avisando de que esta opción no es reversible. 3. El usuario confirma la acción. 4. El sistema borra la hoja seleccionada. Si era la que se estaba viendo por pantalla en ese momento, se pone la vista en otra.

Errores posibles y cursos alternativos	3a. El usuario decide no borrar la hoja y vuelve a donde estaba.

Nombre	**Buscar**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de “Buscar...” o presionando Ctrl+F. 2. El usuario introduce el término o dato a buscar. 3. Las celdas con el término o el dato quedan marcadas.
Errores posibles y cursos alternativos	3a. Si ninguna celda contiene el término o el valor buscado, una alerta avisará al usuario de que no se encuentra ese término en el rango de selección. - Si el usuario utiliza esta función sin haber seleccionado ningún bloque con el caso de uso “Definir Bloque”, la búsqueda se aplicará en toda la hoja. - Si una celda A referencia a una celda B que contiene el valor o término buscado, la celda A será marcada.

Nombre	**Reemplazar**
Actor	Usuario
Precondición	Hay un subconjunto de celdas marcadas con valores coincidentes a la búsqueda del caso de uso “Buscar”
Comportamiento	1. El usuario indica que quiere reemplazar dichos términos.

	2. El usuario introduce el término o valor por el que reemplazar. 3. Pulsando la tecla Enter, el usuario puede reemplazar uno a uno los términos encontrados en el rango de selección.
Errores posibles y cursos alternativos	- El usuario puede utilizar ctrl+Enter para reemplazar todos los términos encontrados en el rango de selección de una vez. - Si el usuario ejecuta la función sin tener un rango de selección activo, se buscará y reemplazará por toda la hoja.

Nombre	**Ordenar**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de “Ordenar Datos”. 2. El usuario selecciona el tipo “Ascendente o Descendente”. 3. El sistema recoloca las filas de forma que queden ordenadas según el valor del contenido de la primera columna. En caso de empate se usan los valores de la columna de la derecha y así sucesivamente.
Errores posibles y cursos alternativos	

Nombre	**Renombrar Hoja [OPCIONAL]**
Actor	Usuario
Comportamiento	1. El usuario hace doble click sobre el nombre de la hoja. 2. El sistema marca el nombre y permite editarlo. 3. El usuario introduce el nombre deseado y confirma los cambios. 4. El sistema modifica el nombre de la hoja.
Errores posibles y cursos alternativos	3a. El nombre no puede quedar vacío ni llamarse igual que alguna otra hoja ya existente.

Nombre	**Añadir Filas**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de añadir fila. 2. El sistema le pregunta cuántas filas desea añadir y en qué posición. 3. El usuario rellena estos campos. 4. El sistema añade el número de filas especificado por el usuario en la posición indicada. 5. El sistema actualiza los IDs de las filas para mantener el orden.
Errores posibles y cursos	3a. Si el usuario introduce un valor erróneo en el número de filas o la posición, el sistema devuelve un error.

alternativos	

Nombre	**Añadir Columnas**
Actor	Usuario
Comportamiento	1. El usuario selecciona la opción de añadir columna. 2. El sistema le pregunta cuántas columnas desea añadir y en qué posición. 3. El usuario rellena estos campos. 4. El sistema añade el número de columnas especificado por el usuario en la posición indicada. 5. El sistema actualiza los IDs de las columnas para mantener el orden.
Errores posibles y cursos alternativos	3a. Si el usuario introduce un valor erróneo en el número de columnas o la posición, el sistema devuelve un error.

Nombre	**Eliminar Filas**
Actor	Usuario
Comportamiento	1. El usuario selecciona una o más filas y posteriormente la opción de eliminar. 2. El sistema pide confirmación por parte del usuario para realizar la acción, avisando que es una acción irreversible. 3. El sistema elimina la(s) fila(s) correspondiente(s). 4. El sistema actualiza los IDs de las filas para mantener el orden.

Errores posibles y cursos alternativos	2a. El usuario cambia de opinión y cancela la acción.

Nombre	**Eliminar Columnas**
Actor	Usuario
Comportamiento	1. El usuario selecciona una o más columnas y posteriormente la opción de eliminar. 2. El sistema pregunta confirmación por parte del usuario para realizar la acción. 3. El sistema elimina la(s) columna(s) correspondiente(s). 4. El sistema actualiza los IDs de las columnas para mantener el orden.
Errores posibles y cursos alternativos	2a. El usuario cambia de opinión y cancela la acción.

Nombre	**Modificar el tamaño de una Fila/Columna [OPCIONAL]**
Actor	Usuario
Comportamiento	1. El usuario hace clic en uno de los bordes laterales de una columna o en los bordes de una fila. 2. El usuario arrastra hacia un lado o hacia el otro para hacer más grande o más pequeña la columna o fila.
Errores posibles y cursos alternativos	1a. Alternativamente, el usuario puede pulsar click derecho sobre una columna o fila, seleccionar la opción de “Tamaño:” y poner un tamaño en píxeles a la columna por valor.

	2a. Una columna o fila no pueden ser más pequeñas que un número determinado de píxeles. Si el usuario intenta hacerlas más pequeñas, la columna o fila se quedará en el valor mínimo establecido.

Nombre	**Definir Bloque**
Actor	Usuario
Comportamiento	1. El usuario hace clic en una celda y mantiene pulsado para marcar el conjunto de celdas que desee. 2. El sistema guarda la posición de la esquina superior izquierda y la esquina inferior derecha del conjunto marcados. 3. El sistema resalta el rectángulo de celdas definido por las esquinas introducidas.
Errores posibles y cursos alternativos	

Nombre	**Copiar Bloque**
Actor	Usuario
Precondición	Hay un bloque de celdas seleccionado con el caso de uso “Definir Bloque”
Comportamiento	1. El usuario selecciona la opción de copiar selección o pulsa Ctrl+C.

	2. El sistema genera un BloqueTemporalCopiado del contenido del bloque definido por el usuario, sobreescribiendo su previo contenido. 3. El sistema marca el bloque original copiado.
Errores posibles y cursos alternativos	

Nombre	**Cortar Bloque**
Actor	Usuario
Precondición	Hay un bloque de celdas seleccionado con el caso de uso “Definir Bloque”
Comportamiento	1. El usuario selecciona la opción de copiar selección o pulsa Ctrl+X. 2. El sistema genera un BloqueTemporalCopiado con el contenido del bloque definido por el usuario. 3. El sistema marca el bloque original cortado en la interfaz de usuario.
Errores posibles y cursos alternativos	

Nombre	**Pegar Bloque**
Actor	Usuario
Comportamiento	1. El usuario selecciona una celda. 2. El usuario selecciona la opción de pegar selección o pulsa Ctrl+V.

	3. El sistema pega el bloque de celdas del BloqueTemporalCopiado, sobrescribe las celdas necesarias, siendo la celda seleccionada la posición de la esquina superior izquierda.
Errores posibles y cursos alternativos	3a. Si el BloqueTemporalCopiado estaba vacío, no se pega ningún competido. 3b. Si no hay espacio en la hoja para pegar el contenido del portapapeles, el sistema le avisará y le preguntará si desea ampliar la hoja hasta que el bloque entero quepa en ella. 3c. Si los datos guardados del BloqueTemporalCopiado provienen de la operación Cortar, cuando el usuario pega la selección, el sistema borra la selección original.

Nombre	**Modificar el contenido de una celda**
Actor	Usuario
Comportamiento	1. El usuario selecciona una única celda. 2. El usuario introduce el nuevo valor que tomará el campo de la celda. 3. El sistema actualiza la información de la celda con la nueva introducida. 4. El usuario selecciona una celda distinta para ver reflejados los cambios o pulsa Enter (definir los valores que puede tener).
Errores posibles y cursos alternativos	2a. Si el usuario introduce un tipo de dato/función con una sintaxis errónea, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra.

Nombre	**Definición de celda por referencia**
Actor	Usuario
Comportamiento	1. El usuario selecciona una única celda. 2. El usuario introduce el carácter clave '=' + el identificador de otra celda. 3. El sistema reemplaza la información de la celda seleccionada con la de la celda asociada a ese identificador.
Errores posibles y cursos alternativos	2a. Si el usuario especifica una celda que contiene una referencia a otra, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra. 2b. Si el usuario especifica una celda que no existe dentro de esa hoja, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra.

Nombre	**Definir Función**
Actor	Usuario
Comportamiento	1. El usuario selecciona una única celda. 2. El usuario introduce el carácter clave '=' y a continuación el nombre de la función que quiere utilizar.

	3. El usuario introduce los valores con los que trabajar (valores inmediatos o referencias a otras celdas). 4. El usuario aprieta Enter para ver el resultado de su función.
Errores posibles y cursos alternativos	2a. Si el usuario introduce una función que no existe, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra. 3a. Si el usuario no introduce suficientes parámetros o introduce parámetros de más, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra. 3b. Si el usuario introduce datos que su tipo no corresponden al tipo pedido en los parámetros, el sistema marcará la celda con un error de tal forma que el usuario se dé cuenta al seleccionar otra.

Nombre	**Formatear color fondo [OPCIONAL]**
Actor	Usuario
Comportamiento	1. El usuario selecciona una celda o un rango de celdas. 2. El usuario selecciona la opción “Color de Celda...”. 3. El usuario selecciona un color. 4. El color de fondo de la celda pasa a ser igual al seleccionado.
Errores posibles y cursos alternativos	

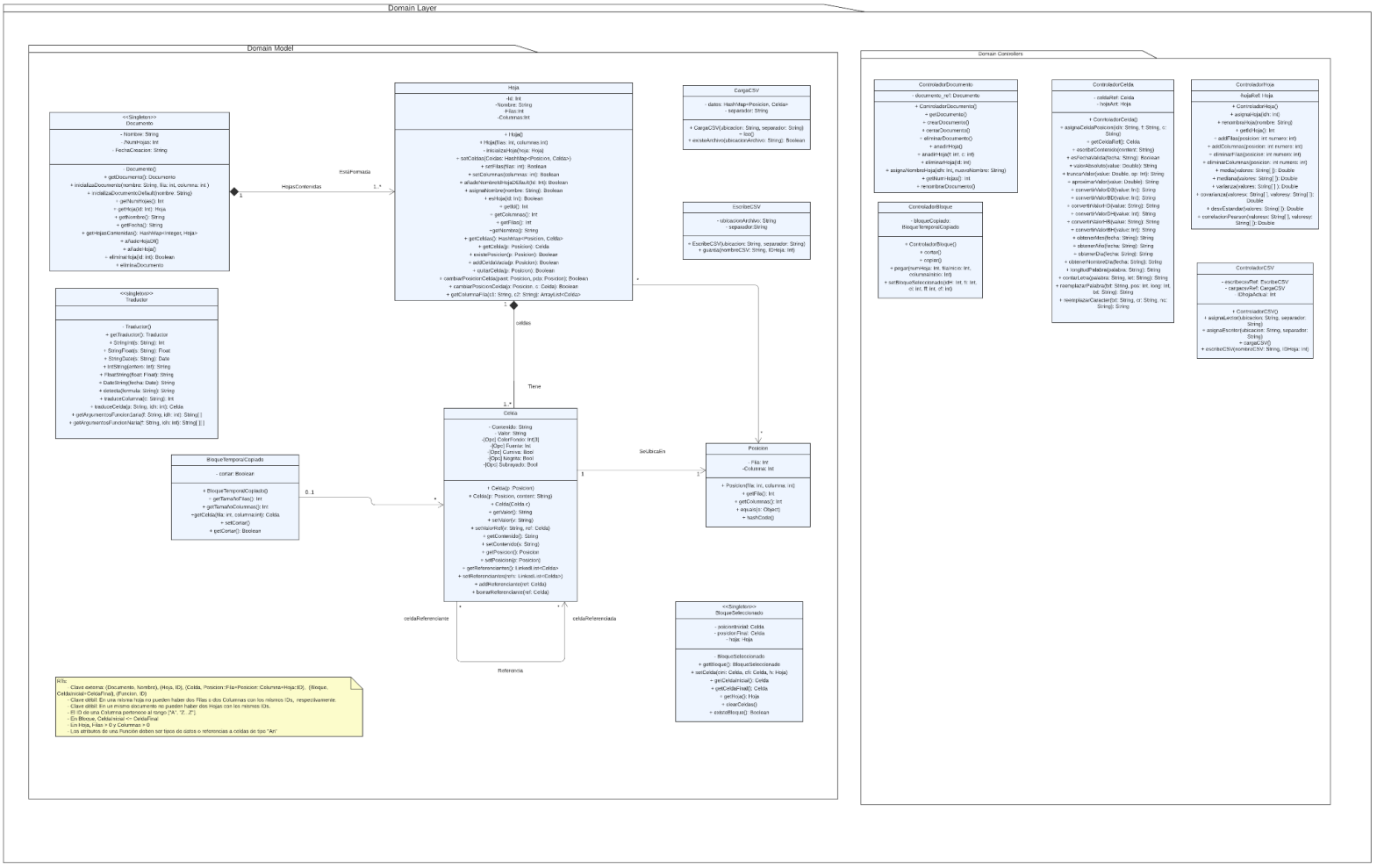
Nombre	**Formatear fuente [OPCIONAL]**
Actor	Usuario
Comportamiento	1. El usuario selecciona una celda o un rango de celdas 2. El usuario selecciona la opción “Fuente de Celda...” 3. El usuario selecciona un color para la fuente, establece su tamaño, alineación, tipo de fuente, subrayado/negrita/cursiva 4. La fuente pasa a tener los parámetros especificados
Errores posibles y cursos alternativos	

Nombre	**Formatear bordes [Opcional]**
Actor	Usuario
Comportamiento	1. El usuario selecciona una celda o un rango de celdas. 2. El usuario selecciona la opción “Formatear Bordes de Celda...”. 3. El usuario selecciona el tipo de borde que quiere aplicar (fino, grueso, malla...) y color del borde. 4. Los bordes del rango de selección pasan a ser del tipo seleccionado.
Errores posibles y cursos alternativos	

2. Diagrama del modelo conceptual de datos

2.1 Diagrama del modelo conceptual

- Dado que aquí no se dispone del tamaño suficiente para mostrar el diagrama correctamente, se puede encontrar una versión ampliada en la misma carpeta de este documento



Restricciones Textuales:

- Claves externas: (Documento, Nombre), (Hoja, Id), (Celda, Posicion::Fila+Posicion::Columna+Hoja::Id), (Bloque, CeldaInicial+CeldaFinal), (Función, Id)

- Clave débil: En una misma hoja no pueden haber dos Filas o dos Columnas con los mismos IDs, respectivamente.

- Clave débil: En un mismo documento no pueden haber dos Hojas con los mismos IDs.

- El ID de una Columna pertenece al rango ["A", "Z...Z"].

- En Bloque, CeldaInicial <= CeldaFinal

- En Hoja, Filas > 0 y Columnas > 0

- Los atributos de una Función deben ser tipos de datos o referencias a celdas de tipo "An"

2.2 Explicación de las clases

Nombre	**Documento**
Descripción	Documento será la clase que contenga todas las hojas creadas por el usuario. Es una clase base, es decir, sin documento, no tiene sentido que exista hojas, celdas, posiciones etc. Además es singleton, por lo que solo podrá haber una instancia de ella.
Cardinalidad	Única
Atributos	- **Nombre** (*str*): Nombre del documento - **FechaCreacion** (*str*): Fecha de creación del Documento - **HojasContenidas** (*HashMap<int, Hoja>*): HashMap de Hojas que contiene el documento - **/NumHojas** (*int*): Número de hojas que tiene el Documento

Nombre	**Hoja**
Descripción	Hoja será la clase que contenga las diferentes celdas y posiciones. El usuario podrá crear tantas hojas como quiera, y podrá asignarle un tamaño. Por defecto consideramos que las hojas tienen un tamaño de 50 filas y 50 columnas.

Cardinalidad	Múltiple
Atributos	- **ID** (*int*): Identificador de la hoja, único para cada una dentro de un mismo documento - **Nombre** (*str*): Nombre de la hoja que aparecerá en la interfaz - **Filas** (*int*): Número de filas que contiene - **Columnas** (*int*): Número de columnas que contiene - **Celdas** (*HashMap<Posicion, Celda>*): HashMap de Celdas existentes en la hoja

Nombre	**Celda**
Descripción	La clase Celda será una unidad de información en una hoja, con valor único asociado. Es donde se guardarán los diferentes valores que el usuario escriba, desde el resultado de funciones hasta texto.
Cardinalidad	Múltiple
Atributos	- **Valor** (*str*): Valor de la celda. Valor mostrado en presentación y enviado a las celdas referenciantes - **Contenido** (*str*): Contenido de la celda. Solo mostrado cuando se edita este. Define el valor, ya sea directamente, mediante una función, una referencia - **Posicion** (*Posicion*): Posición asignada a la celda. - [Opc] **ColorFondo** (*int[3]*): Color en RGB del fondo de la celda - [Opc] **Fuente** (*int*): Familia de fuente que usa el valor mostrado - [Opc] **Cursiva** (*bool*): Determina si el valor se muestra en cursiva - [Opc] **Negrita** (*bool*): Determina si el valor se muestra en negrita - [Opc] **Subrayado** (*bool*): Determina si el valor se muestra subrayado

Nombre	**BloqueSeleccionado**
Descripción	Es una clase que contiene un conjunto seleccionado de i x j

	celdas. Se utilizará principalmente para agrupar las celdas en bloque y poder por ejemplo copiar y pegarlas en otro sitio.
Cardinalidad	Única
Atributos	- **CeldaInicial** (*Celda*): La celda inicial o superior izquierda del bloque seleccionado. - **CeldaFinal** (*Celda*): La celda final o inferior derecha del bloque seleccionado. - **Hoja** (*Hoja*): La hoja en la que se encuentra la selección

Nombre	**Traductor**
Descripción	Clase *singleton* dedicada a la conversión de tipos y detección de argumentos. Esta clase se encarga de traducir las diferentes funciones que hemos creado y de traducir lo que escribe el usuario en las distintas celdas, para que las demás clase puedan trabajar con los datos y ofrecer lo que pide el usuario.
Cardinalidad	Única instancia
Atributos	- **Traductor** (*Traductor*): instancia única de traductor

Nombre	**Posición**
Descripción	Posición *[fila, columna]* en una hoja
Cardinalidad	Única por cada celda en una hoja
Atributos	- **Fila** (*int*): determina la fila - **Columna** (*int*): determina la columna

Nombre	**BloqueTemporalCopiado**
Descripción	Esta clase sirve para poder tener un conjunto de celdas copiadas. Por lo cual solo podrá haber una instancia única de ella.
Cardinalidad	Única
Atributos	- **BloqueCopiado** (*Celda[][]*): Matriz de las celdas copiadas - **Cortar** (*bool*): determina si el bloque ha sido copiado mediante la acción de *cortar*

Nombre	**ControladorBloque**
Descripción	Esta clase será la encargada de manejar las operaciones disponibles que tiene un bloque(cortar, pegar...), así como de gestionar el bloque seleccionado.
Cardinalidad	Única
Atributos	- **BloqueTemporalCopiado (***BloqueTemporalCopiado***)** : determina el bloque temporal copiado

Nombre	**ControladorCelda**
Descripción	Esta clase se encarga de gestionar todo aquello que tiene que ver con celda, se encarga de llamar a las distintas funciones que pide el usuario en una celda, así como de poner su contenido y valores.
Cardinalidad	Única
Atributos	- **HojaAct** (*Hoja*): Determina la hoja en la que esta la celda seleccionada - **CeldaRef** (*Celda*): Determina la celda seleccionada

Nombre	**ControladorDocumento**
Descripción	Es la clase que se encarga de gestionar todo lo relacionado con el documento, desde crearlo, añadir hojas o cambiarle el

	nombre.
Cardinalidad	Única
Atributos	- **Documento_ref** (*Documento*): Se refiere al documento actual abierto

Nombre	**ControladorHoja**
Descripción	Es la clase que se encarga de gestionar todo lo relacionado con una hoja, desde eliminar y añadir filas o columnas a gestionar algunas funciones que dependen de varias celdas e incluso de definir algún bloque de estas.
Cardinalidad	Única
Atributos	- **Hoja_ref** (*Hoja*): Se refiere a la hoja actual en la que estará el usuario

3.Estructuras de Datos y Algoritmos

En este apartado se hablará de las estructuras de datos y algoritmos utilizados y las razones por las cuales se han escogido. Cabe destacar que el proyecto de este año no tiene una fuerte carga de algorítmica en comparación con otros años, por lo que se hará especial hincapié en el subapartado de las estructuras de datos.

Tanto para los algoritmos como las estructuras de datos, se mencionan las clases con información relevante a mencionar.

3.1 Estructuras de Datos

Clase Documento

En la clase Documento utilizamos una tabla de hash (HashMap) para almacenar las hojas. Un documento no suele tener un número muy grande de hojas y por lo tanto la diferencia en eficiencia entre usar una estructura de datos u otra será negligible. Por este motivo, el criterio para elegir una tabla de hash es la simplicidad en la programación al añadir, eliminar y buscar elementos.

Utilizamos un HashMap de Integer - Hoja para almacenar las diferentes hojas que pueda llegar a tener un documento. Estas estarán almacenadas según un entero que corresponderá con su identificador.

Clase Hoja

Esta clase es la que contiene la estructura de datos más importante de todo el programa, que es la encargada del manejo de las celdas de una hoja.

Una celda representa conceptualmente la “unidad básica” de información de una hoja de cálculo. Haciendo la abstracción para un paradigma de programación orientada a objetos, el objeto celda es la base de todo lo demás. Las hojas son

conjuntos de celdas y los documentos conjuntos de hojas (y por lo tanto, conjuntos de conjuntos de celdas).

Esta puntualización se ha hecho para enfatizar la importancia de una correcta elección de la estructura de datos encargada de la gestión de dichas celdas, ya que una hoja puede contener miles, incluso cientos de miles de ellas.

Antes de empezar a pensar en las estructuras de datos candidatas y los pros y contras de unas respecto las otras, es importante matizar para que las vamos a utilizar y cuales son las operaciones que el usuario va a realizar más frecuentemente y menos, para así poder tener un criterio claro para elegir la más óptima.

Identificación de las operaciones comunes

Al trabajar con una hoja de cálculo, la acción que el programa más va a tener que hacer será la de hallar el valor de una celda cualquiera. Esto se usa cuando se pasa una celda como parámetro de una función, cuando el contenido de una celda, del cual dependen otras, cambia y esta tiene que notificar a dichas celdas para que se actualicen en consecuencia, etc. Esto implica que necesitemos un bajo coste en el acceso aleatorio.

También a destacar, pero menos común la iteración de celdas en posiciones contiguas en las hojas de cálculo. En muchos casos, al ser una forma clara y organizada de estructurar la información en una hoja, los valores sobre los que se quiere realizar una función están en filas o columnas contiguas. Por lo tanto, una estructura de datos con un coste bajo en la iteración de elementos funcionaría bien para este tipo de operaciones.

Finalmente, lo que creemos que será una funcionalidad usada poco frecuentemente son las eliminaciones o adiciones ya sea de filas o columnas. Un

usuario estándar suele realizar estas operaciones de manera muy ocasional. Entre estas dos la más habitual sería la inserción de filas o columnas cuando ya ha ocupado todas las disponibles en el momento. Cabe destacar que cuando sucede esto, el usuario puede elegir el número de filas o columnas a insertar. El comportamiento del usuario en un alto porcentaje de estas ocasiones es añadir un número considerablemente grande, para así tener de sobra y no necesitar ir ampliando continuamente. Esto se puede ver en *Google Sheets*[1], donde al llegar al final de una hoja, te permite especificar el número de filas a añadir, dando 1000 como número por defecto como se muestra en la figura X. De esta forma la operación será costosa en cuanto al número de celdas que se añadirán a la hoja, pero que quedará amortizado por las pocas veces que se realizará.

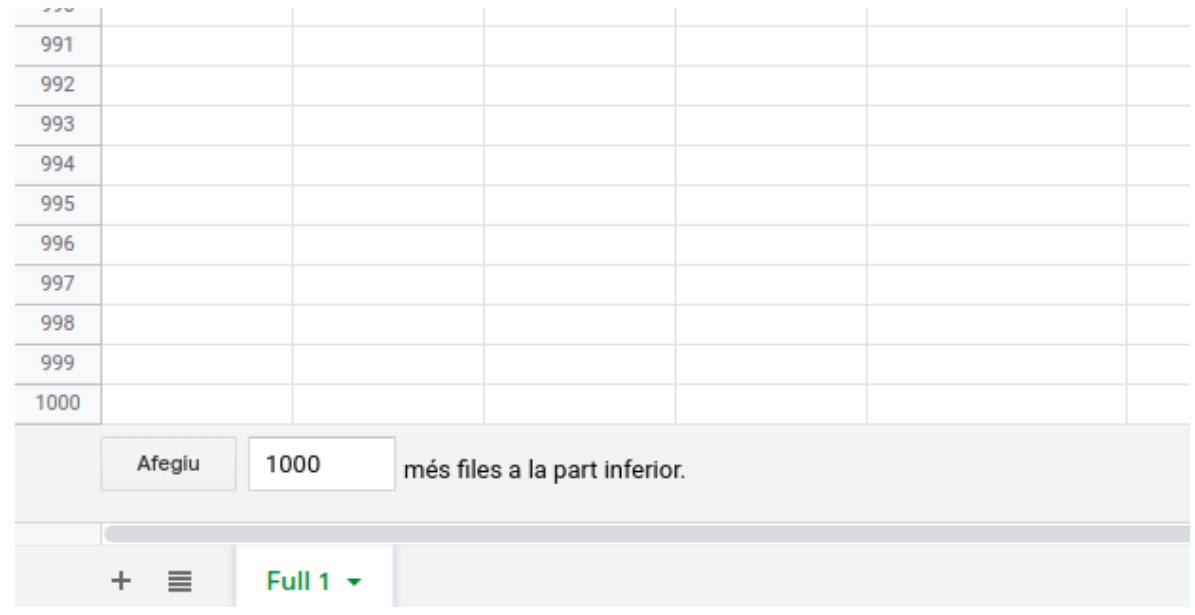


Figura X. Captura de pantalla de la interfaz gráfica para realizar la operación de añadir filas en Google Sheets

Dicho esto, nuestra implementación de una hoja de cálculo permite añadir filas o columnas en la posición que desee el usuario. Es decir, es posible añadir o eliminar bloques de filas o columnas adyacentes. Esto implica que además del coste de procesar las celdas que componen dichos bloques de filas o columnas, se tiene que añadir el coste de actualizar la posición del resto de celdas

afectadas. Un caso donde este hecho cobra mucha importancia sería por ejemplo eliminar la primera fila de una hoja. En este caso habría que sumar al coste de eliminar las celdas de dicha fila, cambiar de posición de todas las celdas (haciendo que su nuevo número de fila sea uno menos). Por lo tanto, por un lado es una funcionalidad poco común y por lo tanto menos relevante en la ecuación, pero por el otro cuando se produce puede llegar a ser muy costosa.

Vistos nuestros requisitos es hora de pasar a ver de qué estructuras de datos disponemos y cual es la que mejor se adapta a lo que queremos hacer.

Comparación de candidatos

Tablas de hash (HashMap<Posición, Celda> en Java):

HashMap es una implementación del concepto de tabla de *hash* en Java. Una tabla de *hash* es una estructura de datos, dado un par <clave, valor>, guarda los elementos en una tabla indexada a partir de una función de *hash*[2] sobre la clave. Los puntos a favor de esta estructura de datos es que permite la inserción, búsqueda y borrado de elementos en tiempo amortizado constante, lo que la hace óptima para los accesos aleatorios (que recordemos que es la operación más usada) y considerablemente buena también para la eliminación y adición de filas o elementos.

Lista de listas (LinkedList<LinkedList<Celda>> en Java):

LinkedList es una implementación del concepto de lista doblemente enlazada en Java. Una lista se caracteriza por guardar los elementos "enlazados", de forma que cada elemento referencia a el elemento anterior y al siguiente y se recorren usando iteradores. Esto exige que exista el concepto de algún tipo de orden entre los elementos. En nuestro caso, este sería claramente el número de fila y columna. El punto a favor de esta estructura de datos es el bajo coste en la iteración sobre sus elementos. Sin embargo, hay que destacar que cada celda tiene dos "dimensiones": número de filas y número de columnas. Esto implica

que en la práctica haya que anidar una lista dentro de la otra de forma que perdemos parte de esa ventaja de una rápida iteración de elementos. Si la lista anidada representa las columnas, la iteración sobre filas contiguas será más rápida que sobre las columnas y viceversa. Otro defecto sería que el número de fila y columna que tiene una celda no se guardaría de forma implícita en la estructura (como sí pasa con un HashMap con la posición como clave o una *array*). Dado un iterador que apunta a un elemento, si no hemos hecho un control anteriormente del número de veces que se ha movido, no tenemos manera de saber el número de fila o columna en el que está, por lo que implica una código más complejo.

Matriz bidimensional (ArrayList<ArrayList<Celda>> en Java):

A*rrayList* es una implementación del concepto de *array*s dinámicas en Java. Una *array* es un conjunto de elementos del mismo tipo guardados de forma contigua en memoria. Cabe destacar que por la manera en que está construido *Java*, los objetos siempre son pasados como referencias, por lo tanto lo que se almacena en dicha *array* son referencias a los objetos. El punto a favor de esta estructura de datos es que el acceso aleatorio a una posición es muy rápido, lo que permite que la selección de elementos y la iteración, las dos operaciones que como hemos mencionado antes los usuarios realizan con más asiduidad, tengan un coste bajo. La gran desventaja viene con las operaciones de borrado de filas y columnas que modifican las dimensiones de la hoja. Estas son muy ineficientes ya que internamente usa una *array* que tiene un tamaño prefijado y para cambiarlo se tiene que crear una nueva y copiar todo el contenido. Esto en la práctica no sucede siempre, ya que la clase A*rrayList*, se encarga de reservar más espacio del solicitado. Aún así por el tipo de uso de la funcionalidad de añadir celdas, donde cuando al usuario se le acaban las filas y columnas pide muchas más de las que necesita, esta acción de tener que crear una nueva *array* y copiar todo el contenido creemos puede suceder con bastante recurrencia, teniendo un alto coste en tiempo de ejecución.

En la siguiente página, se muestra la figura X2 con una tabla resumen de los costes de las operaciones de la hoja de cálculo.

Dada una hoja de n*m celdas:

Estructura de datos	**Buscar elemento**	**Iterar sobre una fila (n celdas)**	**Iterar sobre una columna (m celdas)**
Tabla de hash	O(1) amortizado	O(n)	O(m)
Lista de listas	O(n)+O(m) = O(n+m)	O(n)	n*O(m) = O(n*m)
Matriz bidimensional	O(1)	O(n)	O(m)

Finalmente hemos optado por una tabla de hash con la posición de una celda como clave. Esto nos permite mantener un coste amortizado constante en la operación de buscar un elemento, que es muy importante al ser la acción que más se realizará. Este coste es similar en una matriz bidimensional usando *arrays* y considerablemente más lento en una lista de listas, ya que en estas no existe el acceso aleatorio.

La iteración sobre filas y columnas tiene un bajo coste temporal también en tanto tablas de hash, matrices usando *arrays* y listas de listas, aunque por implementación interna la lista siempre es lo más rápida para una iteración siguiendo un orden.

Finalmente lo más costoso y el mayor punto negativo es el borrado de filas y columnas. Este punto es especialmente negativo con las tres estructuras y al influir distintas variables como la posición en la que se añadan o borren dichas celdas o si es necesario redimensionar el *array* en caso de tratarse de una matriz bidimensional, el coste temporal puede variar. Por ese motivo no se ha añadido a la tabla.

Características del HashMap y complejidad de las operaciones

Se trata de un HashMap de dos clases de objetos propios, una clase Posición, que representa la posición de la celda en la hoja con dos enteros (número de fila y número de columna) y la clase Celda, que contiene toda la información .

Cabe destacar que para poder incluir una clase como clave en un HashMap habrá que modificar el código del *hashCode* y la función *equals* para poder incluir un HashMap con una clave que es un objeto de una clase propia.

```
HashMap<Posicion, Celda> mapa = new HashMap<Posicion, Celda>();
```

Declaración de la estructura de datos

<u>*Selección de una celda:*</u>

La selección de una celda aleatoria tiene coste O(1).

```
Celda cell = mapa.get(new Posicion( f: 1,  c: 2));
```

Ejemplo de obtención de una celda aleatoria

Tanto la inserción, la eliminación, comprobar la existencia de una Celda tienen coste O(1).

```
mapa.put(new Posicion( f: 1,  c: 1), new Celda( v: "A"));
mapa.put(new Posicion( f: 1,  c: 2), new Celda( v: "B"));
mapa.put(new Posicion( f: 2,  c: 1), new Celda( v: "C"));
mapa.put(new Posicion( f: 2,  c: 2), new Celda( v: "D"));
```

Ejemplo de inserción de celdas en 2 x 2 filas y columnas

Insertar una fila: *.put* m veces (número de columnas, O(1)* m) + *.replace* tantas veces como filas haya por debajo de la que se quiere insertar -> O(n).

```
//Insertar una fila
 mapa.put(new Posicion( f: 3,  c: 1), new Celda());
 mapa.put(new Posicion( f: 3,  c: 2), new Celda());
 mapa.replace(new Posicion( f: 3,  c: 1), mapa.get(new Posicion( f: 2,  c: 1)));
 mapa.replace(new Posicion( f: 3,  c: 2), mapa.get(new Posicion( f: 2,  c: 2)));
 mapa.replace(new Posicion( f: 2,  c: 1), new Celda());
 mapa.replace(new Posicion( f: 2,  c: 2), new Celda());
```

Ejemplo de inserción de una fila en especifico (Se inserta la fila 3)

Insertar una columna: *.put* n veces(número de filas, O(1* n)) + *.replace* tantas veces como columnas haya a partir de la que se quiere insertar -> O(n)

Clase Celda

En la clase Celda utilizaremos la estructura de datos LinkedList de Celdas para almacenar todas sus referenciantes. Dado que tendremos que iterar en algunas operaciones sobre todas ellas, e interaccionar con alguna de ellas LinkedList nos proporciona una ventaja en tiempo de recorrido respecto a sus contrincante el ArrayList, dado que el ArrayList es más rápido para el almacenamiento de datos, LinkedList lo es más para la manipulación de estos.

Clase Bloque Temporal Copiado

En la clase BloqueTemporalCopiado se utiliza un Array de Array de Celdas, este tiene la función de almacenar las celdas que temporalmente se han copiado, para desplazarlas o pegarlas en otras posiciones. Como vemos al no ser necesario el acceso a datos, simplemente se utiliza como almacén, y como conocemos el tamaño de antemano, lo más eficaz es utilizar la estructura de Arrays enfrente a otras, ya que dado a su tamaño fijo, esta es más rápida que otras, véase el ejemplo de su prima cercana ArrayList, al ser esta dinámica es algo más lenta.

Controlador Celda

En la clase ControladorCelda utilizaremos diferentes tipos de estructuras de datos para implementar alguna de sus funcionalidades, estas son principalmente Vectores y HashMap.

- Función obtenerMes:
    - ***Vector de String*** donde se guardan los nombres de los meses del año para asociarlos a su número extraído de la fecha.
    - ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida para asociarlo con el vector de meses y poder mostrar el correspondiente.
- Función obtenerAño:
    - ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida y poder mostrar la parte correspondiente.
- Función obtenerDia:
    - ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida y poder mostrar la parte correspondiente.

- Función obtenerNombreDia:

    · ***Vector de String*** donde se guardan los nombres de los días de la semana en castellano.

    · ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida.

    · ***HashMap de Integer - String*** para traducir los nombres de la semana en inglés al castellano.

    · ***Vector de Month*** donde se guardan todos los nombres de los meses para poder usar las funcionalidades de LocalDate y asociarlo al número de la fecha introducida

Clase Traductor

- Funcion getArgumentosFuncion1aria

    · **ArrayList<Celda>** donde se guardan las distintas celdas de un intervalo para seguidamente poder obtener sus valores.

    · **Vector String** utilizado para guardar los valores pedido por una función

- Funcion getArgumentosFuncionNaria

    · **ArrayList<Celda>** donde se vuelve a utilizar para guardar las distintas celdas de un intervalo para seguidamente poder obtener sus valores.

    · **ArrayList<String>** utilizado para guardar los valores pedido por una función

Clase ControladorHoja

- Funciones Estadisticas
-

· **Vector String,** en todas ellas se utiliza como parámetro de entrada donde vienen almacenados todos los valores numéricos para la correcta operación de la función.

3.2 Algoritmos

Dado que este proyecto no tiene una fuerte carga de algoritmos, explicaremos los que consideramos más importantes para el proyecto.

Asignar valor Celda

Cuando el usuario escribe un contenido en una celda, este tiene que ser entendido por nuestro sistema, de tal manera que sepa si está escribiendo una función o simplemente es un texto llano.

Para ello utilizaremos la función detecta de Traductor que nos devolverá qué tipo de contenido ha escrito el usuario y en caso de ser una función que tipo de función pide.

Mediante un switch ejecutaremos las diferentes posibilidades que puede haber escrito el usuario, dentro de cada caso se pedirán los atributos necesarios para la función a realizar a Traductor y se revisará que sean correctos, seguidamente se ejecutarán las funciones necesarias de cálculo de números, estadístico o de texto y se dejará el resultado del valor en la celda que se ha seleccionado, si por algún motivo ocurriera algún error, este deja en el valor de la celda un código de error en vez de un valor normal.

GetColumnaFila

Este algoritmo se encarga de devolver todas las celdas en filas o columnas definidas por una celda inicial y una celda final.

Dependiendo de la posición en que son pasadas la celda inicial y la celda final, tendremos 4 posibilidades de selección. En caso de fila, de derecha a izquierda o de izquierda a derecha y en caso de columna de arriba a abajo o de abajo a arriba. Esto lo sabremos mirando si la posición de la celda inicial es mayor o menor a la de la celda final.

Según lo explicado anteriormente se ejecutara un bucle (hay 4 distintos, 1 por caso), recogiendo las celdas de la estructura de datos comprendidas entre las posiciones pasadas como parámetro.

Detecta

Detecta será el algoritmo que nos diga que está escribiendo el usuario para que nuestro sistema pueda trabajar con ello. Se compone de un switch, donde en caso de función se separará el string hasta el primer paréntesis y se devolverá un string codificado con el lenguaje interno del sistema.

En caso de encontrar una referencia, esto lo sabrá porque seguid del igual encontrara un '$', devolverá el código #REFERENCIA para comunicar al sistema de que el usuario está pidiendo una referencia.

Finalmente si este no detecta ningun tipo de función o referencia devuelve #VALUE para indicar que lo que está escribiendo el usuario es texto llano
Si por algún motivo ocurriera un error este devolverá #ERROR FUNC

Traduce Celda

Este algoritmo servirá para que dado una posición de la forma 'A1' se obtenga la celda que de verdad hay en esa posición. Dado que nuestro sistema trabaja con posiciones numéricas, hará falta una traducción de la posición. Una vez traducida la posición mediante una función del propio Traductor, este podrá obtener la celda que se encuentra en la posición pasada como texto y la devolverá.

GetArgumentosFuncion1aria

El algoritmo que recoge y devuelve los argumentos de una función 1aria a partir de un string con la función y sus parámetros.

Primero comprueba que el string que llega es una función, y que no se ha colado una string que pueda causar problemas al programa. Crea una lista, llamada “ret”, donde se guardarán los argumentos necesarios. Si los argumentos contienen una referencia simple, añade el valor de esa referencia como parámetro. Si contienen una referencia compleja ($A1:$C1) , por cada celda en el rango, se añade su valor como parámetro. Si no contiene ninguna de estas, se añade directamente como valor a la lista de parámetros. Finalmente, devuelve la lista de argumentos.

GetArgumentosFuncionNaria

El algoritmo que recoge y devuelve los argumentos de una función 1aria a partir de un string con la función y sus parámetros.

Crea una lista de arrays de Strings, y separa los parámetros que llegan. Por cada uno de estos, comprueba si es una referencia compleja ($A1:$C1) o simple, o si es un valor directo. Añade los argumentos correspondientes a la lista y la devuelve.

4. Relación de las clases implementadas por cada miembro del equipo

4.1 Relación de Clases

David Pérez	**Iván Risueño**	**Quim Torra**	**Dani Gallardo**
Hoja	Traductor	Celda	ControladorHoja
ControladorCelda	ControladorBloque	ControladorDocumento	Posicion
Documento	BloqueSeleccionado		BloqueTemporalCopiado

4.2 Relación de Drivers y Test

Nota: Quien ha hecho el driver o test, también ha realizado los Stubs, Mocks, juegos de prueba y explicaciones

David Pérez	**Iván Risueño**	**Quim Torra**	**Dani Gallardo**
ControladorCelda Driver	DriverControladorBloque	DriverDocumento	BloqueTemporalCopiado Test
HojaTest	MainTest	TestCelda	PosicionTest
DocumentoTest	TraductorTest		
	BloqueSeleccionadoTest		
	TestSuite		
	TestRunner		