Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

CE2103: Algoritmos y Estructuras de Datos II

# Proyecto #3

TEC-MFS


**Estudiantes**

Guillermo Sánchez Cedeño

2024132555

Sofia Rubí González

2024211034

**Profesor**

Leonardo Araya Martínez

# INTRODUCCIÓN

TEC Media File System es un proyecto enfocado en imitar el funcionamiento de las arquitecturas RAID5 que se utilizan a nivel profesional para almacenar datos de alta importancia dentro de empresas o personas que necesitan una gran cantidad de espacio y rendimiento de lectura.

El RAID 5 se basa en 3 o más discos conectados a un controlador quien es el encargado de distribuir los datos a través de las unidades activas y, en caso de una pérdida de información, reconstruyen la información.

Este proyecto nos incita abiertamente a investigar acerca de las distintas arquitecturas que se utilizan en la vida real para la productividad o protección de datos mediante la paridad de los discos duros, que permite la recuperación de los datos en caso de que una unidad de almacenamiento falle, que, tarde o temprano, lo van a hacer, es por ello que existe el RAID.

# ÍNDICE

## Descripción del problema

En nuestra solución imitaremos el funcionamiento del controlador de un RAID y los nodos conectados a él (discos duros), usando un servidor principal y pequeñas de instancias que corran vinculadas a carpetas del sistema de archivos de la computadora, además aunado a ello desarrollaremos una aplicación con interfaz gráfica que permita al usuario administrar los datos que se guardan en el RAID.

La aplicación está pensada para correr en múltiples computadoras de manera remota utilizando gRPC, como puente de comunicación entre las distintas instancias activas y el servidor controlador, en donde la interfaz gráfica se comunica con el controlador y este en el fondo coordina los intercambios de información con los nodos registrados en el RAID.

El NodeController se encargará de coordinar a los DiskNodes, de lo que se encargará es de almacenar las referencias en donde se guardan las partes de cada archivo subido al sistema y después cuando el usuario lo solicite. reconstruir la información, además de esto, gestionará la paridad para asegurar que, en caso del fallo de algún DiskNode, se puedan recuperar los datos y brinde una seguridad excepcional.

# Descripción de la solución

## Requerimiento 001

Creamos 2 proyectos separados para crear dos entidades con una arquitectura híbrida, hablamos del ControllerNode y los DiskNodes, quiénes actuarán tanto como un cliente así también como un servidor. En el caso del ControllerNode este necesitará recuperar la información guardada en bloques de los DiskNodes (por lo que realizará peticiones) y estos últimos tienen que darse a conocer con el controlador registrando sus datos como dirección IP o puerto asociado mediante una llamada al servidor principal.

Usamos tinyxml2, una librería externa, para poder leer los archivos de configuración XML e implementando una clase aplicamos la información obtenida de estos archivos. Entre estos datos se incluye el puerto, IP, ID de Nodo, entre otra información relevante para el controlador.

## Requerimiento 002

Cada archivo que llega al sistema se divide en múltiples bloques que posteriormente se distribuyen entre los nodos (discos) disponibles del RAID. Como parte de la distribución, calculamos el bloque de paridad utilizando la operación XOR, lo que permite aplicar la tolerancia a fallos característica del RAID 5.

Encapsulamos toda esta lógica en la clase NodeController. Esto nos permitió mantener el código del servidor mucho más limpio, limitando su responsabilidad únicamente a atender peticiones externas y delegar el trabajo interno al controlador.

Al iniciar el ControllerNode habilita una dirección IP y un puerto específico que permiten establecer conexiones tanto con los nodos de almacenamiento como con la interfaz gráfica.

## Requerimiento 003

Se desarrolló una interfaz gRPC que permite realizar operaciones de carga, eliminación, búsqueda por nombre y descarga de los documentos almacenados en el RAID. Se definió en la clase FileSystemServiceImpl y está responde a las peticiones externas haciendo uso de métodos, cuya lógica se encuentran en el NodeController, como AddDocument, GetDocument, DeleteDocument y GetDocumentList.

Cuando el usuario agrega un documento, por medio de la interfaz gráfica, este se divide en bloques y se distribuye en los distintos nodos. Para la descarga, se recuperan los bloques del archivo y si algún nodo genera un fallo es sistema reconstruye el bloque perdido utilizando la información de paridad. Además, se implementó una lógica de búsqueda que permite ubicar documentos mediante su nombre, permitiendo que el cliente reciba una respuesta sobre la existencia o no del archivo en el RAID.

## Requerimiento 004

Para completar la funcionalidad del ControllerNode, se integró un método adicional dentro de su interfaz gRPC que permite consultar el estado general del sistema RAID. Esta funcionalidad permite consultar información sobre cada uno de los nodos registrados.

Cada vez que un nodo de disco se registra en el sistema, su información es almacenada, lo que permite que pueda ser consultada posteriormente, esa funcionalidad fue implementada con el método GetSystemStatus.

## Requerimiento 005

La aplicación con interfaz gráfica se implementó utilizando las herramientas que Python, nos brinda, el cual es un lenguaje de programación muy fácil de utilizar y para nuestro caso es más que suficiente. Decimos que es lo justo y necesario para nuestro caso debido a que el mismo se utilizará en la capa más visible de la solución, es decir que este no tiene la responsabilidad de realizar cálculos pesados u operaciones costosas que se le podrían complicar, este trabajo lo cedemos directamente a C++.

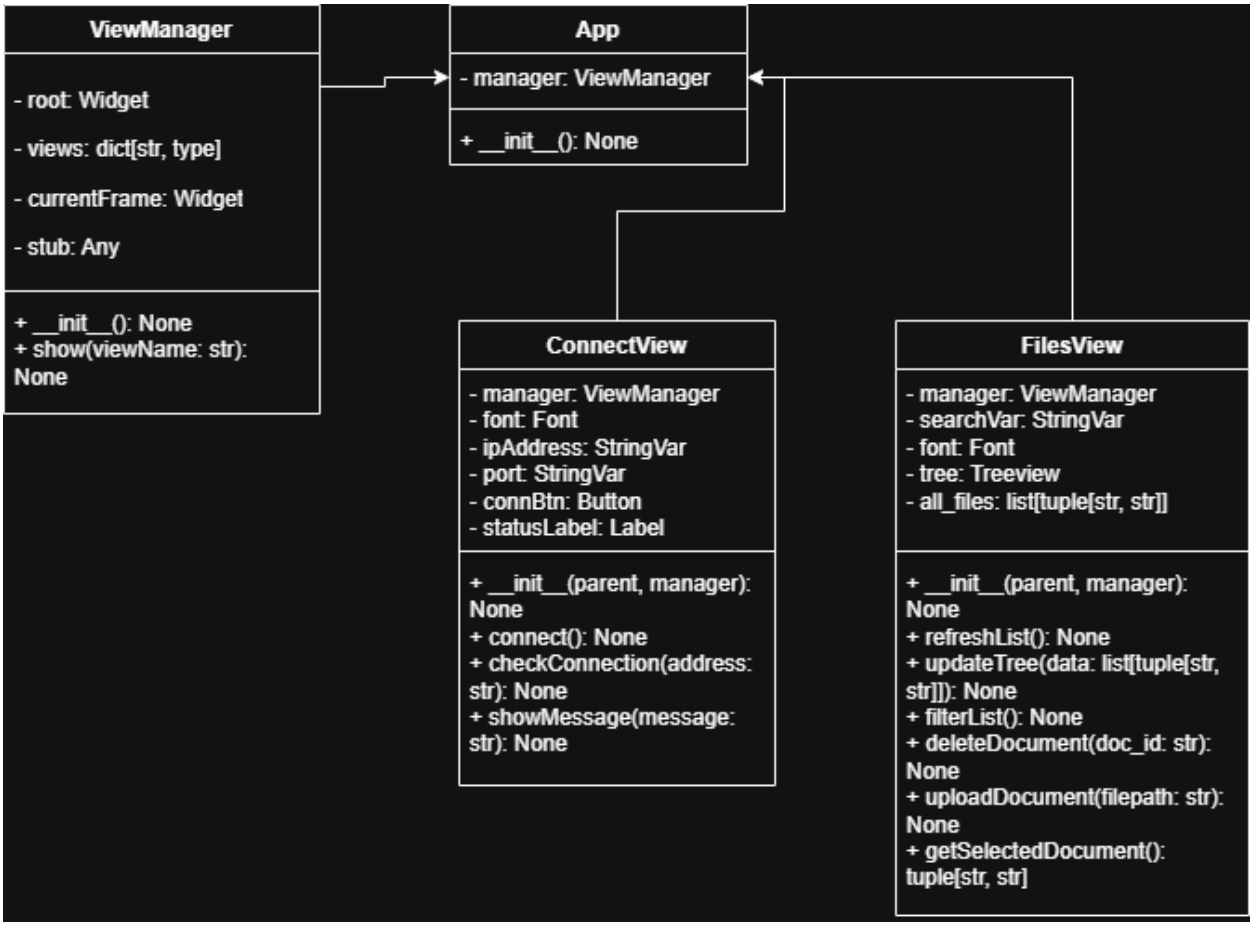
Usamos la librería tkinter para construir la ventana y ttkbootstrap para darle un aire más moderno y evitar el uso de los widgets anticuados que incluye la herramienta por defecto, también se instaló la versión pythonica de gRPC y Protobuf para realizar la comunicación entre la aplicación gráfica y el ControllerNode.

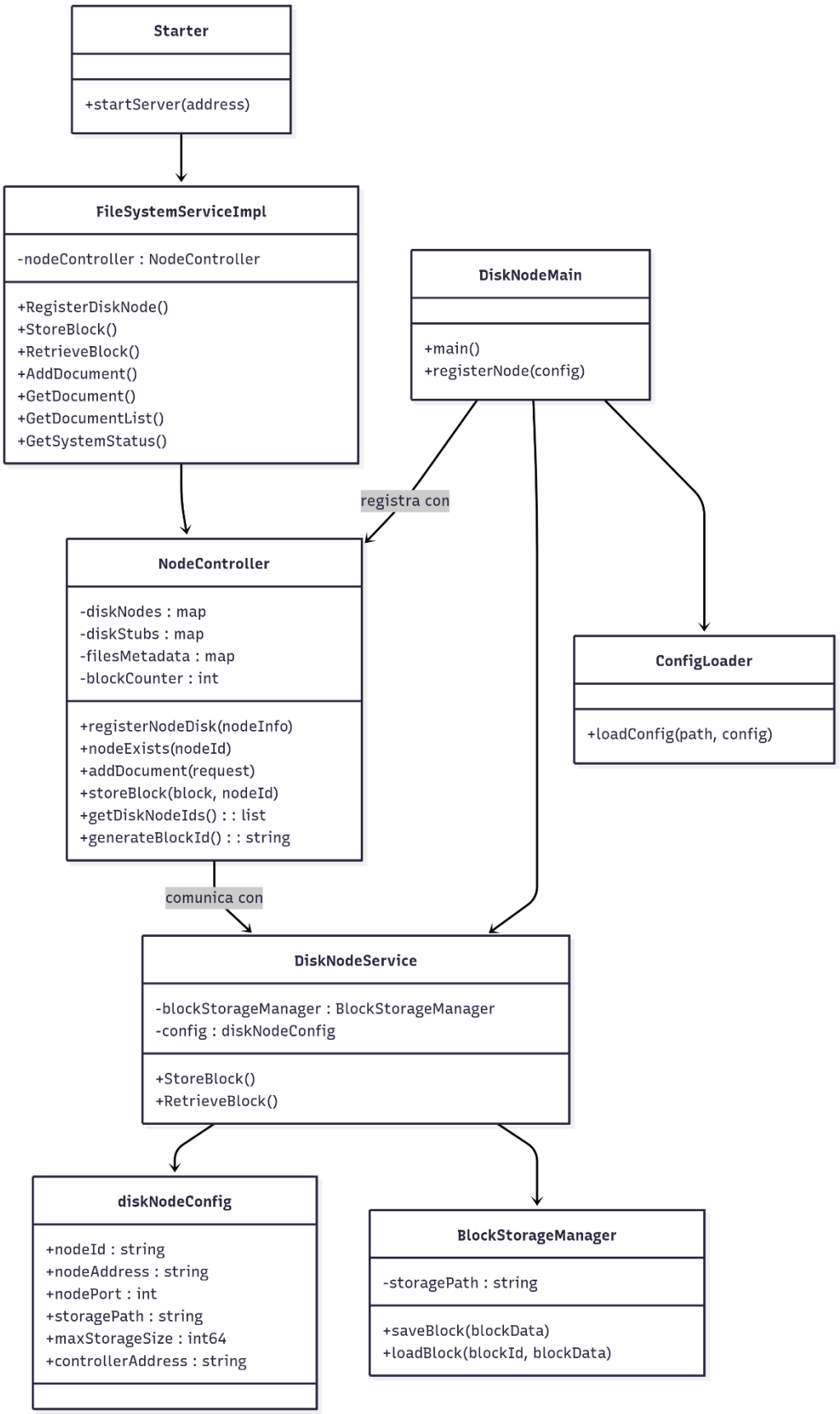
Como punto inicial el usuario tendrá una ventana para ingresar tanto dirección IP como el puerto asociado, después de recolectados estos datos iniciales, realizaremos una validación que compruebe que, en efecto, tenemos conexión al servidor indicado, una vez eso termine la aplicación cambiará de vista, en donde encontraremos un panel bastante sencillo, en donde tendremos varios controles y la lista de documentos almacenados dentro del RAID.

Parte de los controles que se le brindan al usuario es la capacidad de buscar entre la lista de documentos disponibles, cargar nuevos documentos al RAID o borrar documentos innecesarios, además se brinda de pequeños metadatos como la fecha de subida, el peso del archivo y otros datos que al usuario pueda interesar.

En primera instancia esta interfaz gráfica no se iba a realizar en este lenguaje de programación (Python), lo que pensábamos es integrarlo directamente utilizando C# y WPF (Windows Presentation Foundation) pero después de intentar programarlo, nos dimos cuenta de que añadía un peso de complejidad al proyecto que, en consideración del tiempo límite de entrega, no estábamos dispuestos a aceptar. Es por ello, se decidió a utilizar algo mucho más sencillo y dado a las necesidades mínimas de la aplicación, Python era la mejor opción.

# Diseño general




Starter
+startServer(address)
FileSystemServiceImpl
-nodeController : NodeController
+RegisterDiskNode()
+StoreBlock()
+RetrieveBlock()
+AddDocument()
+GetDocument()
+GetDocumentList()
+GetSystemStatus()
DiskNodeMain
+main()
+registerNode(config)
registra con
NodeController
-diskNodes : map
-diskStubs : map
-filesMetadata : map
-blockCounter : int
+registerNodeDisk(nodeInfo)
+nodeExists(nodeId)
+addDocument(request)
+storeBlock(block, nodeId)
+getDiskNodeIds() : : list
+generateBlockId() : : string
ConfigLoader
+loadConfig(path, config)
comunica con
DiskNodeService
-blockStorageManager : BlockStorageManager
-config : diskNodeConfig
+StoreBlock()
+RetrieveBlock()
diskNodeConfig
+nodeId : string
+nodeAddress : string
+nodePort : int
+storagePath : string
+maxStorageSize : int64
+controllerAddress : string
BlockStorageManager
-storagePath : string
+saveBlock(blockData)
+loadBlock(blockId, blockData)

# Repositorio de GitHub

*https://github.com/Mandamientos/tec-media-file-system*