Autoevaluación

1. Porcentaje de cumplimiento

Se cumplió con la totalidad del proyecto (100%) con todos los requerimientos, tanto funcionales, como no funcionales.

2. Información general de diseño de alto nivel, arquitectura, patrones, mejores prácticas utilizadas

El sistema implementado es un Sistema de Archivos Distribuido por Bloques (DFS) inspirado en arquitecturas como HDFS y GFS. El diseño busca garantizar alta disponibilidad, tolerancia a fallos, escalabilidad y eficiencia en la gestión de archivos distribuidos. A continuación, se detallan los principales aspectos de diseño, arquitectura, patrones y buenas prácticas:

Diseño de alto nivel

- **Componentes principales:**
- **NameNode:** Nodo central encargado de la gestión de metadatos, estructura de directorios, mapeo de archivos a bloques y coordinación de la replicación. Implementa alta disponibilidad mediante un esquema Leader-Follower.
- **DataNodes:** Nodos de almacenamiento responsables de guardar los bloques de datos, replicar bloques y reportar su estado al NameNode. Utilizan gRPC para la transferencia eficiente de datos.
- **Cliente CLI:** Interfaz de línea de comandos que permite a los usuarios interactuar con el sistema (subida, descarga, gestión de archivos y directorios).

Arquitectura

- **Separación de canales:**
    - **Canal de control:** Comunicación REST entre Cliente y NameNode para operaciones de metadatos y control.
    - **Canal de datos:** Comunicación gRPC entre Cliente y DataNodes (y entre DataNodes) para transferencia y replicación de bloques.
- **Replicación y tolerancia a fallos:**
    - Cada bloque se almacena en al menos dos DataNodes (Leader y Follower) para garantizar disponibilidad ante fallos.
    - Heartbeats periódicos de DataNodes al NameNode para detección de nodos caídos.
    - Failover automático del NameNode Follower en caso de caída del líder.

- **Particionamiento:**
  - Los archivos se dividen en bloques de tamaño configurable, permitiendo paralelismo y distribución eficiente.
- **Persistencia:**
  - Los metadatos se almacenan en una base de datos SQLite en el NameNode.

Patrones y mejores prácticas utilizadas

- **Modularidad:** Separación clara de responsabilidades entre NameNode, DataNode, Cliente y código compartido.
- **Inyección de dependencias:** Uso de dependencias en FastAPI para facilitar pruebas y mantenimiento.
- **Validación de datos:** Uso de Pydantic para asegurar la integridad de los datos en las APIs.
- **Documentación automática:** Exposición de la API REST con Swagger UI y ReDoc.
- **Paralelismo:** Uso de ThreadPoolExecutor para operaciones concurrentes de subida/descarga de bloques.
- **Configurabilidad:** Parámetros de ejecución configurables por la línea de comandos y variables de entorno.
- **Pruebas automatizadas:** Scripts y módulos de pruebas para verificar la funcionalidad de los componentes principales y comandos CLI.
- **Manejo de errores y logs:** Manejo centralizado de errores y logging estructurado para facilitar el monitoreo y depuración.

Resumen visual de la arquitectura

```
[Cliente CLI] <--REST--> [NameNode Leader] <--gRPC--> [NameNode Follower]
     |                          |
     |                          v
     +--------------------> [DataNode 1] <--gRPC--> [DataNode 2] ...
[DataNode N]
```

Esta arquitectura permite escalar horizontalmente el almacenamiento, soportar fallos de nodos individuales y mantener la integridad y disponibilidad de los datos.

3. Descripción del ambiente de desarrollo y técnico

A. ¿Cómo se compila y ejecuta?

- Inicialmente se necesita instalar las dependencias necesarias con `pip install -r requirements.txt`.
- Seguidamente abrimos una terminal para ejecutar un NameNode con: `python -m src.namenode.api.main --id namenode1 --host localhost --rest- port 8000 --grpc-port 50051`

- Para una configuración robusta con replicación, se recomienda iniciar al menos 3 DataNodes. Abre una terminal nueva para cada DataNode y ejecuta:

DataNode 1:

```
python -m src.datanode.main --node-id datanode1 --hostname localhost --
port 7001 --storage-dir ./data/datanode1 --namenode-url
http://localhost:8000
```

DataNode 2:

```
python -m src.datanode.main --node-id datanode2 --hostname localhost --
port 7002 --storage-dir ./data/datanode2 --namenode-url
http://localhost:8000
```

DataNode 1:

```
python -m src.datanode.main --node-id datanode3 --hostname localhost --
port 7003 --storage-dir ./data/datanode3 --namenode-url
http://localhost:8000
```

- Finalmente necesitamos ejecutar el Cliente CLI:

```
python -m src.client.cli --namenode http://localhost:8000
```

- Para información más detallada se recomienda visitar la **Guía de Ejecución del Sistema** en la documentación adjunta.

B. Detalles del desarrollo

- **Arquitectura:**
- NameNode (gestión de metadatos, API REST, alta disponibilidad Leader-Follower)
- DataNode (almacenamiento de bloques, gRPC, replicación Leader-Follower)
- Cliente CLI (interfaz de comandos para usuarios)
- **Patrones:**
- Separación de canales de control (REST) y datos (gRPC)
- Replicación y tolerancia a fallos
- Modularidad y separación de responsabilidades

C. Detalles técnicos

- **Librerías principales:**
    - FastAPI (API REST)
    - gRPC (transferencia de bloques)
    - SQLAlchemy y SQLite (metadatos)

    - Pydantic (validación de datos)
- **Protocolo de comunicación:**
    - REST para control y metadatos
    - gRPC para transferencia de bloques y replicación
- **Base de datos:**
    - SQLite para metadatos en NameNode
- **Configuración:**
    - Parámetros por la línea de comandos para puertos, IDs, rutas de almacenamiento, etc.
    - Se encuentran dentro del archivo `requirements.txt`

D. Descripción y configuración de parámetros

- **NameNode:**
- `--id`: Identificador único
- `--host`: Dirección de escucha
- `--rest-port`: Puerto REST
- `--grpc-port`: Puerto gRPC
- `--known-nodes`: Lista de otros NameNodes (alta disponibilidad)
- **DataNode:**
- `--node-id`: Identificador único
- `--hostname`: Dirección de escucha
- `--port`: Puerto gRPC
- `--storage-dir`: Carpeta local para bloques
- `--namenode-url`: URL del NameNode
- **Cliente CLI:**
- `--namenode`: URL del NameNode
- `--block-size`: Tamaño de bloque personalizado

E. Organización del código (estructura de directorios)

```
DFS_Bloques/
├── src/
│   ├── namenode/       # NameNode y API REST
│   ├── datanode/       # DataNode y servicios gRPC
│   ├── client/         # Cliente CLI
│   ├── common/         # Código y proto compartido
├── data/               # Almacenamiento de bloques y metadatos
├── tests/              # Pruebas automatizadas y scripts de arranque
├── requirements.txt    # Dependencias
├── README.md           # Documentación principal
└── Documentación/      # Documentos y resúmenes técnicos
```

4. Referencias

https://www.youtube.com/watch?v=WB37L7PjI5k
https://www.youtube.com/watch?v=g6VWTEtUsQY

https://www.youtube.com/watch?v=YklG9N*Qsok*
https://www.youtube.com/watch?v=f3YQbsX1xQ4
*https://medium.com/illumination/tktk-design-a-file-system-with-python-5bd0551ae45d https://www.youtube.com/watch?v=GMae*Gva1dM
https://www.oreilly.com/library/view/hadoop-with-python/9781492048435/ch01.html
https://www.nexsoftsys.com/articles/beginners-tutorial-for-hadoop-file-system-with-python.html