Manual Técnico - Proyecto 2 Manejo E implementación de Archivos

## **Información General**

Henry Alexander García Montúfar

201407049

Manejo e Implementación de Archivos

Vacaciones del primer semestre 2025

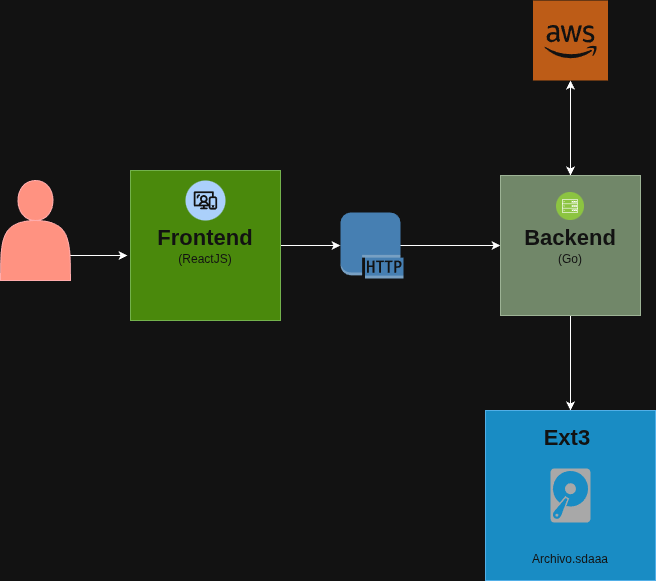
## **Descripción de la Arquitectura del Sistema**

El sistema implementado se apoya en una arquitectura de tipo cliente-servidor, compuesta por distintos módulos distribuidos en la nube, cada uno cumpliendo una función específica y complementaria:

* **Frontend (ReactJS):** Se trata de una interfaz gráfica desarrollada con ReactJS y desplegada en un bucket de Amazon S3, lo que permite su acceso como sitio web estático. Esta interfaz proporciona a los usuarios una experiencia interactiva, permitiéndoles enviar comandos, visualizar los resultados de sus operaciones, explorar de manera gráfica la estructura del sistema de archivos EXT3 y acceder al registro de operaciones realizado mediante journaling.
* **Backend (Go):** Consiste en una API RESTful desarrollada en el lenguaje Go y alojada en una instancia EC2 con sistema operativo Ubuntu. Su función principal es recibir las solicitudes enviadas desde el frontend, interpretar los comandos y ejecutarlos sobre archivos binarios con extensión .sdaa, que representan discos virtuales con formato EXT3. Adicionalmente, el backend gestiona procesos de recuperación de información a través del journaling, garantizando integridad y trazabilidad de las operaciones.

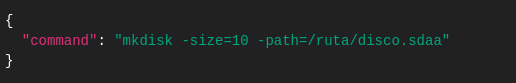
Ambos módulos están interconectados mediante solicitudes HTTP bajo el paradigma REST, lo que facilita una comunicación eficiente y desacoplada entre la interfaz de usuario y la lógica del servidor.

### **Diagrama de Arquitectura**



### **Backend en Go (API REST)**

La lógica del servidor está implementada en **Go**, utilizando el framework web **Fiber**, el cual permite un manejo eficiente y rápido de solicitudes HTTP. Se expone un endpoint principal (/execute), que acepta peticiones **POST** con un payload en formato **JSON**, estructurado de la siguiente manera:



#### **Flujo de ejecución del backend:**

1. Recibe el comando enviado por el cliente a través de una solicitud POST.
2. El comando es analizado e interpretado por un componente denominado **Analyzer**, encargado de descomponer y validar las instrucciones.
3. Una vez validado, se ejecuta la acción correspondiente sobre archivos con extensión .sdaa, los cuales representan discos simulados.
4. Finalmente, se genera una respuesta que es enviada al frontend, incluyendo información del resultado o estado del sistema.

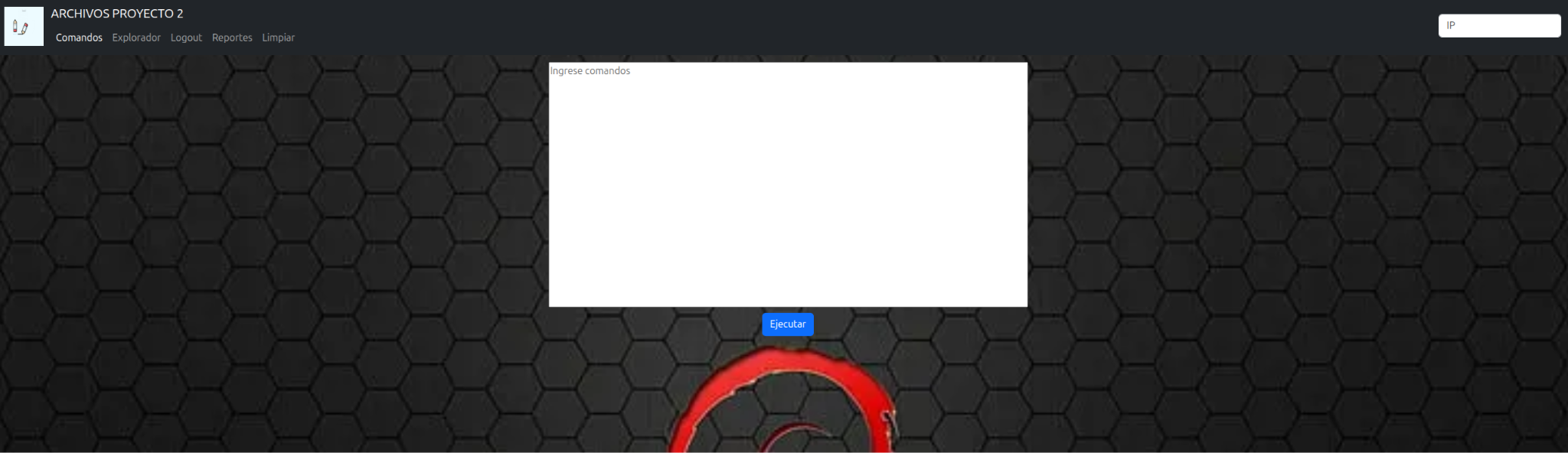
El servidor escucha en el **puerto 3000** y tiene habilitado **CORS**, permitiendo la comunicación entre dominios para aceptar peticiones provenientes del cliente web.

### **Frontend en React**

La interfaz gráfica del sistema está desarrollada con **React** y es desplegada mediante **Amazon S3**, facilitando un acceso rápido y escalable desde el navegador.

#### **Funcionalidades del frontend:**

* Permite al usuario **ingresar comandos personalizados** para interactuar con el sistema de archivos simulado.
* Incluye funciones de **inicio y cierre de sesión** para controlar el acceso de usuarios.
* Ofrece una **visualización gráfica en modo de solo lectura** del sistema de archivos, permitiendo explorar su estructura jerárquica.
* Proporciona opciones para **mostrar el contenido de archivos y visualizar el journaling**, con el fin de auditar operaciones realizadas sobre el sistema.



### **Explicación de las Estructuras de Datos**

#### **MBR (Master Boot Record)**

El **MBR** es una estructura fundamental ubicada en el primer sector del disco. Contiene información crítica sobre la partición del disco, incluyendo los metadatos que definen hasta **cuatro particiones primarias** o **una partición extendida**. Esta estructura permite al sistema operativo identificar cómo está organizado el almacenamiento físico desde el arranque.

#### **EBR (Extended Boot Record)**

El **EBR** es una estructura auxiliar utilizada exclusivamente dentro de las particiones extendidas. Sirve como **cabecera de cada partición lógica** y permite el **encadenamiento** de múltiples particiones lógicas dentro de una extendida, mediante una estructura de lista enlazada.

#### **Superbloque**

En los sistemas de archivos del tipo **EXT3**, el **superbloque** es una estructura que almacena información esencial sobre el sistema de archivos, incluyendo:

* Número total de **inodos** y **bloques**.
* Ubicación de los **bitmaps**.
* Parámetros de montaje.
* Estado del sistema de archivos.
* Referencias al sistema de **journaling**.  
   Esta estructura es clave para validar y montar correctamente un sistema de archivos.

#### **Inodos**

Los **inodos** representan cada archivo o directorio del sistema. No contienen directamente el nombre del archivo, pero sí una serie de **metadatos** esenciales:

* Tipo de archivo (directorio, archivo regular, enlace, etc.).
* Permisos de acceso.
* Fechas de creación, modificación y acceso.
* Punteros a los bloques de datos asociados.

#### **Bloques**

El sistema EXT3 organiza los datos en diferentes tipos de bloques:

* **Folder Blocks**: almacenan referencias a directorios e inodos hijos.
* **File Blocks**: almacenan referencias a archivos.
* **Content Blocks**: contienen el contenido real de los archivos.

EXT3 genera aproximadamente **tres veces más bloques que inodos**, debido a la segmentación lógica y la necesidad de representar adecuadamente tanto la jerarquía como los datos.

#### **Bitmaps**

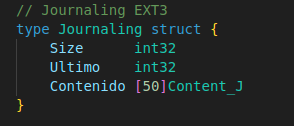
* **Bitmap de inodos**: es una estructura que indica qué inodos están actualmente en uso.
* **Bitmap de bloques**: cumple la misma función para los bloques de datos.  
   Ambos bitmaps permiten una gestión eficiente del espacio disponible en el sistema de archivos.

#### **Journal (Registro de transacciones)**

Una de las mejoras más destacadas de EXT3 respecto a sus predecesores es la incorporación del **journaling**.  
 El **Journal** es una nueva estructura que registra todas las operaciones críticas antes de aplicarlas al sistema de archivos, incluyendo:

* Tipo de operación (creación, modificación, eliminación).
* Usuario que ejecuta la acción.
* Ruta del archivo o directorio afectado.
* Contenido involucrado.
* Marca temporal de la operación.

Esto garantiza una mayor integridad y confiabilidad del sistema ante fallos inesperados, permitiendo recuperar el estado anterior sin comprometer los datos.



### **Comandos Implementados**

El sistema cuenta con un conjunto robusto de comandos para gestionar discos virtuales y simular operaciones reales sobre sistemas de archivos tipo EXT3. A continuación se describen los comandos principales:

* **mkdisk** Genera un disco virtual nuevo con su respectivo **MBR** (Master Boot Record), asignándole tamaño y ruta específica con extensión .sdaa.
* **fdisk** Permite la **creación, eliminación o redimensionamiento** de particiones dentro de un disco virtual. Admite las opciones -add para agregar espacio y -delete para eliminar particiones existentes.
* **mount** Monta una partición previamente creada y le asigna un **ID único** en memoria, que se utilizará para todas las operaciones subsecuentes sobre esa unidad.
* **mkfs** Realiza el formateo de la partición montada, inicializando la estructura del sistema de archivos EXT2 o EXT3 (por defecto). Se encarga de generar el **superbloque**, los **bitmaps**, la tabla de inodos y el archivo inicial users.txt.
* **login** Permite el inicio de sesión dentro de una partición montada mediante un usuario y contraseña válidos definidos en users.txt.
* **mkdir** Crea uno o varios directorios dentro del sistema de archivos.
* **mkfile** Genera archivos, opcionalmente con contenido inicial, dentro del sistema de archivos.
* **cat** Muestra en consola el contenido de un archivo especificado.
* **unmount** Desmonta una partición, liberando su ID y cerrando el acceso a su contenido.
* **journaling** Visualiza gráficamente el contenido del journal, permitiendo auditar operaciones realizadas (sin exportarse en formato .dot).

### **Visualización Web del Sistema de Archivos**

Se ha desarrollado una interfaz web que permite la interacción visual con el sistema de archivos montado:

#### **Ejemplos de Uso de Comandos:**

* **mkdisk**

****

Crea un disco virtual y genera automáticamente un MBR inicial.

* **fdisk**

****

**Agrega una partición al disco virtual, pudiendo ser primaria, extendida o lógica.**

* **mount**

****

**Monta la partición especificada y la asocia a un ID interno único.**

* **mkfs**

****

**Formatea la partición como EXT3, generando todas las estructuras necesarias: superbloque, bitmaps, inodos, y el archivo users.txt.**

* **login**

****

**Autentica al usuario administrador dentro de la partición montada.**

### **Visualización Web del Sistema de Archivos**

**La interfaz web incluye las siguientes características de visualización:**

* **Selector de disco y partición montada.**
* **Representación gráfica jerárquica en modo de solo lectura de archivos y carpetas.**
* **Visualización directa del contenido de archivos .txt.**
* **Panel interactivo de auditoría para examinar el contenido del journal.**

### **Pruebas de Pérdida y Recuperación**

**Se han diseñado pruebas que simulan fallos en el sistema para validar los mecanismos de restauración:**

* **Simulación de pérdida con loss  
   Elimina intencionalmente entradas de bitmaps, bloques de contenido e inodos activos.**
* **Recuperación con recovery  
   Reconstruye la estructura de archivos y carpetas utilizando la información registrada en el journal y el superbloque.**
* **Validación  
   Las pruebas confirman que, tras la recuperación, las carpetas, archivos y permisos son restaurados correctamente, validando la robustez del sistema ante fallos críticos.**

### **Despliegue en AWS**

**El sistema ha sido desplegado exitosamente en la nube utilizando Amazon Web Services (AWS):**

#### **Frontend (ReactJS)**

* **Compilado con npm run build.**
* **Publicado en un bucket de Amazon S3 con acceso público.**
* **Configurado para servir index.html como entrada principal.**

#### 

#### 

#### **Backend (Go)**

* **Ejecutado en una instancia EC2 (Ubuntu Server).**
* **Backend iniciado manualmente con go run main.go o como servicio persistente.**
* **El puerto 3000 habilitado en el Security Group.**
* **CORS configurado para aceptar solicitudes desde dominios cruzados (frontend).**

### **Conclusión**

**Este proyecto extiende un sistema de archivos simulado tipo EXT2 hacia una implementación más avanzada compatible con EXT3, incorporando manejo realista de bitácoras (journaling), capacidades de recuperación ante pérdidas, y una interfaz web intuitiva. El despliegue en AWS permitió validar el sistema en un entorno distribuido, promoviendo la integración entre frontend y backend, y asegurando la persistencia y visualización de datos binarios en la nube.**