Ejercicio: Consola Unix

Se deberá implementar en Javascript lo siguiente:

Modelar una estructura de archivos estilo Unix, junto con ciertos comandos del sistema operativo que permiten manipular a los archivos.

El sistema de archivos debe permitir tener archivos y directorios. Existe un directorio destacado que es el raíz. Cada directorio puede contener archivos, u otros directorios. El usuario actual puede estar posicionado en un directorio a la vez. Inicialmente es el raíz, pero puede ir a un subdirectorio usando el comando `cd`. Para volver al directorio padre, debe usar el comando `cd ..`. Para ver el contenido del directorio actual, puede usar el comando `ls` (que va a retornar todos los archivos y subdirectorios que haya).

Además existen usuarios, que tienen un nombre y una contraseña. Existe un usuario administrador conocido como root.

El sistema debe contemplar lo siguiente:

- Iniciar sesión como root o como cualquier usuario. Se debe lanzar un error si el usuario no existe, o la contraseña es incorrecta.
- Cada vez que se inicia sesión, el usuario debe comenzar en el directorio raíz.
- Agregar o remover usuarios (sólo root puede hacerlo). El nombre de usuario sólo puede contener letras minúsculas, números y guiones (-_). La longitud debe ser entre 6 y 25 caracteres.
- Crear y borrar archivos y directorios en el directorio actual.
- El archivo tiene un contenido textual que puede ser editado en cualquier momento. Nos interesa saber si un usuario puede editar un archivo o no.
- Navegar entre los diferentes directorios (tal como lo hace el comando `cd`)
- Listar el contenido de un directorio (tal como lo hace el comando `ls`)
- Administrar los permisos de archivos y directorios:
    - Tenemos permisos de lectura, escritura y ejecución (para los archivos) y de lectura y escritura (para los directorios)
    - A cada archivo se le asignan permisos para el usuario *owner* (o sea, el usuario logueado que creó el archivo), y para los demás.
    - Los permisos deben ser chequeados al momento de una operación.
    - El usuario root puede ejecutar cualquier operación con los archivos.
    - El owner de un archivo o directorio siempre tiene permisos de lectura y escritura.
    - Ejemplo: si el archivo `tesis.txt` es propiedad de Juan, y tiene permiso sólo de lectura para los demás, entonces Juan (y root) pueden borrar el archivo (porque ambos tienen permiso de escritura), pero el usuario Pedro sólo puede leer su contenido (porque es de los “demás”)

Segunda parte

1. Agregar las operaciones `mv` y `cp`, que mueven y copian, respectivamente, archivos y/o directorios. Utilizar los permisos existentes para validar estas operaciones.
2. Poder registrar "aplicaciones" asociadas para abrir ciertos tipos de archivos. Por ejemplo, los archivos con extensiones `.txt` y `.csv` se abren con `vim`. De las aplicaciones sólo se necesita saber el nombre. No hace falta hacer la lógica para abrir el archivo, sólo saber si tiene una aplicación asociada. Cuando no haya una aplicación asociada, debe avisar al usuario con un mensaje.
3. Implementar links simbólicos (lo que hace el comando `ln -s`).
4. Implementar búsqueda por texto en archivos. Por ejemplo: buscar todos los archivos que tengan "Pepe" en su contenido, dentro de la carpeta donde estamos posicionados (es posible que haya resultados en sub-carpetas, también deben ser incluidos). La búsqueda debe ser *case-insensitive* (es decir, sin tener en cuenta mayúsculas y minúsculas).
5. Hacer un intercambio de permisos entre dos usuarios. La idea es que cada usuario tenga ahora también los del otro usuario. Por supuesto si el usuario A tenía acceso a un archivo, pero el usuario B no tenía, al final el usuario A debe seguir teniendo acceso a ese archivo.
6. Implementar modo binario en los archivos. Esto quiere decir, que cuando se escriben, su contenido es sólo unos y ceros. Pero al leerlo, se debe transformar ese contenido a texto. Cuando creamos un archivo, queremos que por defecto sea textual, pero podemos indicar si queremos que sea binario.
7. Implementar la posibilidad de enviar el resultado del comando `ls` a un archivo en el disco real (como lo hace el operador >). La salida esperada tiene la siguiente estructura:

```
miDirectorio:
      miSubDirectorio     lectura                         juan
      miArchivo.txt       lectura/escritura               root
      miArchivo2.txt      lectura/escritura/ejecución     pepe
```