# Documentación desafío Backend “Consola” ShipNow

## INFORMACION:

Realizado por Nicolás Francisco Garilli.

Lenguaje Utilizado: Java.

– Utilice openjdk-20 para realizar el desafío.

## INTRODUCCIÓN:

El motivo de este documento es transmitir cómo fui modelando las entidades para llevar a cabo el enunciado, y explicar porque tome las decisiones que tome a la hora de modelar el mismo. Además, de mencionar algunas cuestiones que me parecen relevantes.

### File y Folder, IResource.

Para iniciar, modele las clases File y Folder. Encontré que se podía aplicar el patrón de diseño composite ya que una Folder puede tener dentro de sí misma tanto Files como otras Folders, que a su vez, estas Folders pueden contener dentro suyo otras Folders.

Además, aplicar este patrón de diseño me permite tratar tanto a los Files como Folders de manera uniforme, ya que ambas deben de implementar la Interfaz IResource.

Con respecto a la metadata decidí que sea la fecha de creación, y el contenido puede ser cualquier string.

### FileSystem.

Básicamente es el sistema de archivos que nos dice cual es la carpeta en la que estamos actualmente además de crear la carpeta raíz ‘root’.

### Consola.

Con la primer parte del enunciado me encontré que era muy fácil repetir muchas veces codigo, por lo cual implemente funciones como validateArguments() para validar que la cantidad de argumentos que recibe las operaciones de la consola sean las indicadas, y así reutilizar esta función en los comandos que necesiten esta validación.

Posteriormente realice lo mismo agregando la función loggedIn() para chequear todos los

comandos que requieran saber si el usuario estaba logueado.
Las funciones showFile() y showMetadata() en un principio eran casi iguales y se repetía muchisimo código, por lo cual decidí crear la función showResource() para evitar tanto código repetido, y dependiendo el parámetro que recibe muestra el content o la metadata.
También agregue el comando ‘exit’ para poder cerrar la consola desde la terminal.

## Usuarios

Con respecto a las operaciones que puede realizar un User decidí que sean Crear Carpetas, Crear Archivos y Destruirlos. De esta manera se puede seguir navegando por la consola y ver el contenido de las carpetas y los archivos sin necesidad de iniciar sesión.
Además de esto agregue el comando ‘logout’ para cerrar la sesión que me parecía una buena implementación.
También pensé en agregar un tipo de encriptación para la password de los Users, ya que tenerlo como Strings no es seguro, pero para realizar esto requiriera importar librerías externas y el enunciado prohibia esto. “La consola no puede utilizar gemas o librerías externas al lenguaje”

## Espacios de trabajo.

Para realizar este enunciado, me parecio lo mas optimo hacer que las clases que quiero almacenar implementen la interfaz “Serializable” lo cual me permite persistir el sistema de archivos con todas las Folders y Files creadas, tanto como con los Users creados en un archivo llamado filesystem.ser en la carpeta del proyecto.
Esta solución me parecía la más sencilla de implementar para el problema que plantea el enunciado, ya que no requiere el uso de librerías externas a java.
En algún momento pensé en convertirlos a formato JSON y quizás almacenarlos en alguna base de datos documental, pero decidí que para este enunciado una solución más sencilla era la apropiada.
Para agregar, me costó comprender si este enunciado pedía que el archivo a persistir iba a ser uno único el cual tenía que recibir por parámetro o si la consola tenía que poder ser inicializada en otro modo en el cual persista todo lo que creemos desde que la iniciamos.
Termine por realizar que todo lo que hayas creado desde el inicio de la sesión se persista cuando realizas el comando ‘exit’ para salir de la consola, y la próxima vez que vuelvas a iniciarla, automáticamente te carga todo lo que hayas creado la sesión de uso anterior.

## **Tests**:

Los tests fueron quizá la parte que me llevó más tiempo del enunciado, ya que la mayoría de las funcionalidades implementadas en el desafío son Prints en consola, realizar tests exhaustivos sobre estos puede llegar a ser un tanto incómodos. Para funcionalidades más simples quizá es más cómodo simplemente utilizar la consola y ver si funciona como se esperaba. Sin embargo, realice varios tests probando las funcionalidades más críticas y complejas.

## **Nota Final:**

El enunciado me entretuvo ya que nunca había realizado una actividad similar, me hubiese gustado implementarla en Ruby para familiarizarme con el lenguaje, pero desafortunadamente me encuentro en semana de finales a la par que realizaba este enunciado.

No quería demorar en entregarlo, ya que estaba entusiasmado por completarlo.

Gracias por darme la oportunidad de realizar este desafío, espero que mi solución les agrade.