TP III - SO

**Ideia Geral**

* Desenvolvimento em uma linguagem OO, de forma mais simples possível pra não gastar muito tempo. Importante ter uma interface amigável com operações de arquivos, porque o TP vai ser basicamente a abstração disso. Sugestão: **Python**.
* Utilização de um frontend web, utilizando-se de plugins e frameworks pra agilizar o máximo possível. Sugestão: **JSX, React, Pug**.
* Desenvolvimento de uma pequena camada de API RESTful pra fazer a comunicação do frontend e backend. Sugestão: **Flask**.
* Utilizar-se de containers docker pra facilmente instalar e subir a aplicação.
* Subir a aplicação no servidor dedicado ([gfviegas.com](http://gfviegas.com))

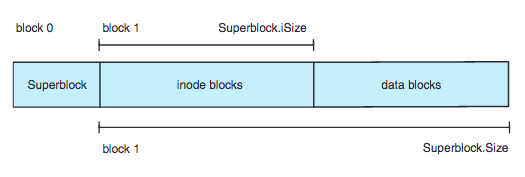
Backend

* Seguir o modelo do link a seguir para as operações a serem criadas: <https://www.os-book.com/OS9/java-dir/12.pdf>
* Utilização de algum plugin pra logging, onde podemos facilmente implementar um modo verboso e logar na tela ou disparar pra um arquivo as operações realizadas.

Frontend

* Interface de interação com o usuário dos parâmetros inicias do sistema, como tamanho do bloco, etc. Utilização de sliders para limitar esses dados, etc.
* Visualização dos arquivos como tree-views (Exemplo: <https://diogofcunha.github.io/react-virtualized-tree/#/examples/basic-tree>)
* As operações sobre um diretório poderiam ser feitas simplesmente com o uso de botões de ações + dialog com caminho pra mover diretório, ou nome de um arquivo, por exemplo.
* Upload de arquivo em formato que devemos definir com as operações para manipulação de arquivos/diretórios. Que tal usarmos o mesmo padrão do Unix?
* Extra: Visualização do disco, blocos.

Estrutura do File System

O sistema de arquivos é composto por um **superblock** que descreve o status do sistema de arquivos. Internamente, cada arquivo é armazenado no sistema utilizando uma estrutura de dados de *inode*.

Disco Simulado

Pra simular um disco, é usado uma estrutura chamada **DISK** que simula um disco com **NUM\_BLOCKS** blocos de **BLOCK\_SIZE** bytes por bloco. As constantes em vermelho são definidas pelo usuário ao iniciar uma simulação (dado um certo intervalo)

**NUM\_BLOCKS** deve pertencer ao intervalo (X0, X1).  
**BLOCK\_SIZE** deve pertencer ao intervalo (Y0, Y1).

Dessa forma, **DISK** possui **BLOCK\_SIZE x NUM\_BLOCKS** bytes e pode ser interpretada como nossa primeira classe (Ver diagrama de classe). As operações que podemos fazer sobre um disco são: ler o n-ésimo bloco, escrever no n-ésimo bloco.  
Ao iniciar **DISK**, se cria um UID da simulação, e se cria um arquivo com nome UID, de **BLOCK\_SIZE x NUM\_BLOCKS** bytes, ocupados por diversos X. Deve-se persistir esse UID, em um cache Redis, pra persistir simulações. Após fazer tudo isso, ele cria o superblock, que é o bloco 0.

Cada bloco deve ocupar sempre exatamente **BLOCK\_SIZE** bytes. Ou seja, todas as operações de leitura e/ou escrita de um bloco deve retornar exatamente essa quantidade de bytes. Um bloco nada mais é que uma estrutura.

**SUPERBLOCK** contém informações sobre o disco como: número de blocos totais, número de blocos de inodes, índice do primeiro bloco vazio.

**INODES** são armazenados a partir do bloco 1 até **SUPERBLOCK.**númeroDeBlocosDeInodes. O inode armazena o seu tamanho em bytes, flags, tamanho do arquivo que representa e 10 índices diretos pra blocos do disco e 1 índice indireto. Dessa forma teremos que o tamanho máximo do arquivo, será de **(BLOCK\_SIZE / 4 + 10)** **\* BLOCK\_SIZE** bytes. Exemplo: **BLOCK\_SIZE = 4KB**, teremos tamanho máximo do arquivo de 4.23MB. (CONSULTAR DANIEL SE ISSO É SUFICIENTE)

Cada bloco de inode possui **BLOCK\_SIZE / tamanho Inode** quantidade de inodes.

**DIRETORIO, ARQUIVO**

**DIRETORIO** contém uma lista de itens