Documentação Básica do Código

Este código implementa um servidor Node.js que processa imagens enviadas pelo usuário.

Bibliotecas utilizadas:

- express: framework web para Node.js
- express-fileupload: middleware para gerenciar uploads de arquivos
- path: módulo para manipulação de caminhos de arquivos
- fs: módulo para operações no sistema de arquivos
- performance-now: módulo para medir tempo de execução (opcional)
- worker_threads: módulo para utilização de workers (threads)

Funcionamento geral:

1. O servidor é iniciado na porta 5502.
2. O cliente acessa a rota `/` e recebe o arquivo `login.html`.
3. Ao enviar imagens pela rota `/home-pag`, o servidor:
    - Registra o recebimento da requisição.
    - Define o diretório de saída para os arquivos processados.
    - Recupera os arquivos de imagem enviados.
    - Inicia a contagem do tempo de processamento.
    - Para cada imagem enviada:
        - Cria um worker thread para processar a imagem individualmente.
        - Envia os dados da imagem e o caminho de saída para o worker.
    - Aguarda a finalização de todos os workers.
    - Calcula o tempo total de processamento.
    - Define cabeçalhos para evitar cache das imagens processadas.
    - Envia uma resposta contendo os URLs das imagens processadas (`/output/outputN.jpg`).
    - Após 1 minuto, exclui todos os arquivos processados da pasta de saída.
4. Em caso de erro durante o processamento, o servidor retorna um erro (código 500) para o cliente.

Estrutura do código:

- **Declaração de dependências:** Importa as bibliotecas necessárias.
- **Configuração do servidor Express:**
    - Define o middleware para upload de arquivos.
    - Define rotas para servir arquivos estáticos:
        - `/`: serve o arquivo `login.html`.
        - `/output`: serve arquivos da pasta `output`.
- **Rota `/home-pag`:**
    - Processa o envio de imagens:
        - Recupera os arquivos enviados.
        - Inicia medição de tempo.
        - Utiliza workers para processamento paralelo.

        - Calcula o tempo total e envia resposta com URLs das imagens processadas.
        - Remove arquivos temporários após um tempo configurado.
    - Trata erros e envia mensagem apropriada para o cliente.
- **Iniciação do servidor:** O servidor escuta na porta 5502.

Workers:

- **Declarações de importação:**
    - `parentPort`: esta variável vem do `worker_threads`módulo e fornece um canal de comunicação entre o trabalhador e o thread principal.
    - `workerData`: esta variável também vem `worker_threads`e contém os dados passados ao trabalhador a partir do thread principal.
    - `sharp`: Esta linha importa a biblioteca Sharp para processamento de imagens.
- **Exploração madeireira:**
    - `console.log('passei aqui')`: esta linha simplesmente imprime uma mensagem no console indicando que o trabalhador iniciou o processamento.
- **Processamento de imagem:**
    - `sharp(workerData.fileData)`: Esta linha cria um objeto Sharp a partir dos dados da imagem (`fileData`) recebidos no `workerData`thread principal.
    - `.greyscale()`: Este método converte a imagem em tons de cinza.
    - `.toFile(workerData.output)`: Este método salva a imagem processada em tons de cinza no caminho de saída especificado (`workerData.output`) também fornecido no arquivo `workerData`.
- **Comunicação com thread principal:**
    - `.then(() => { ... })`: Este bloco lida com o processamento de imagem bem-sucedido.
        - `parentPort.postMessage({ success: true })`: Esta linha envia uma mensagem de volta ao thread principal indicando o `parentPort`processamento bem-sucedido com uma `success`propriedade definida como `true`.
    - `.catch((err) => { ... })`: este bloco trata quaisquer erros que ocorram durante o processamento.
        - `parentPort.postMessage({ success: false, error: err })`: esta linha envia uma mensagem de erro de volta ao thread principal. A mensagem inclui uma `success`propriedade definida como `false`e uma `error`propriedade contendo os detalhes do erro.