Tipo de Armazenamento

**Assunto:** Adaptar o Armazenamento de Dados de um Projeto Web para um Aplicativo Desktop Tauri

Objetivo:

O objetivo desta tarefa é adaptar o sistema de armazenamento de dados do meu projeto de aplicação web, que atualmente utiliza localStorage, para um sistema de arquivos local, preparando-o para ser compilado como um aplicativo de desktop com o Tauri. As funcionalidades de salvar, carregar, exportar (backup) e importar (restaurar) dados devem ser modificadas para interagir com o sistema de arquivos do usuário através de um backend Rust (Tauri), em vez do armazenamento do navegador.

**Situação Atual do Projeto:**

1. **Armazenamento Principal:** Os dados são salvos e carregados usando as funções saveJSON e loadJSON no arquivo C-utilitarios.js, que manipulam o localStorage do navegador.
2. **Gerenciamento de Estado:** O arquivo B-armazenamento-dados.js utiliza essas funções para persistir o estado da aplicação. A mutação LOAD\_STATE é síncrona.
3. **Backup e Restauração:** A funcionalidade de "Exportar Dados" (exportAllData) e "Importar Dados" (importAllData) está no arquivo R-modulo-configuracoes.js. Atualmente, a exportação cria um arquivo blob e simula um download, enquanto a importação usa um <input type="file"> para ler um arquivo JSON.

**Arquitetura Alvo (Tauri):**

O aplicativo será compilado com o Tauri. Isso nos permite usar um backend em Rust para interagir com o sistema de arquivos do usuário. O JavaScript do frontend se comunicará com o Rust através do comando invoke do Tauri.

**Premissa:** Assuma que as seguintes funções já foram criadas no backend Rust e expostas para o frontend. Você não precisa escrever o código Rust, apenas o JavaScript que as chama:

* save\_data\_to\_file(data: String): Recebe uma string JSON e a salva em um arquivo de dados padrão do aplicativo.
* load\_data\_from\_file() -> String: Lê o arquivo de dados padrão e retorna seu conteúdo como uma string JSON.
* save\_backup\_file(data: String): Abre uma janela de diálogo "Salvar Como..." nativa do sistema operacional e salva a string JSON fornecida no local escolhido pelo usuário.
* restore\_backup\_file() -> String: Abre uma janela de diálogo "Abrir Arquivo..." nativa, permite que o usuário selecione um arquivo .json, e retorna o conteúdo desse arquivo como uma string JSON.

**Instruções Detalhadas de Modificação:**

**Passo 1: Modificar o Arquivo C-utilitarios.js**

Este é o passo central. As funções saveJSON e loadJSON precisam parar de usar localStorage e começar a invocar as funções do backend Rust. Elas se tornarão assíncronas.

* **Arquivo a ser modificado:** C-utilitarios.js
* **Funções a serem alteradas:** saveJSON e loadJSON.

**Alteração na saveJSON:**

* A função deve ser transformada em async.
* Deve usar window.\_\_TAURI\_\_.invoke('save\_data\_to\_file', { data: JSON.stringify(data) }).

**Alteração na loadJSON:**

* A função deve ser transformada em async.
* Deve chamar window.\_\_TAURI\_\_.invoke('load\_data\_from\_file').
* Como a função Rust retorna uma string, o resultado precisa ser convertido de volta para um objeto com JSON.parse. Se a chamada falhar ou retornar vazio, deve retornar o valor fallback.

**Passo 2: Modificar o Arquivo B-armazenamento-dados.js**

Como a função loadJSON agora é assíncrona, a mutação LOAD\_STATE que a chama também precisa se tornar assíncrona para usar await.

* **Arquivo a ser modificado:** B-armazenamento-dados.js
* **Função a ser alterada:** A mutação LOAD\_STATE dentro do objeto store.mutations.

**Alteração na LOAD\_STATE:**

* Declare a função como async: async LOAD\_STATE(state) { ... }.
* Use await ao chamar loadJSON: const loadedEscalas = await loadJSON(KEYS.escalas, []);.
* Faça o mesmo para todos os outros loadJSON dentro desta função (funcionarios, turnos, cargos, etc.).

**Passo 3: Modificar o Arquivo R-modulo-configuracoes.js**

As funções de backup e restauração devem parar de simular downloads/uploads e começar a usar as funções Rust que abrem diálogos nativos do sistema.

* **Arquivo a ser modificado:** R-modulo-configuracoes.js
* **Funções a ser alteradas:** exportAllData e importAllData.

**Alteração na exportAllData:**

* A função deve ser transformada em async.
* A lógica de coletar todos os dados do localStorage deve ser mantida.
* Em vez de criar um Blob e um link de download, a função deve converter o objeto de dados em uma string JSON e passá-la para a chamada window.\_\_TAURI\_\_.invoke('save\_backup\_file', { data: dataStr }).

**Alteração na importAllData:**

* A função deve ser transformada em async.
* Remova a criação do elemento <input type="file">.
* Chame window.\_\_TAURI\_\_.invoke('restore\_backup\_file') para abrir o diálogo de arquivo e obter o conteúdo.
* O resto da lógica (confirmar com o usuário, parsear o JSON, salvar os dados e recarregar a página) deve ser adaptado para usar a string retornada pelo invoke.

**Restrições:**

* A lógica de manipulação de dados dentro do B-armazenamento-dados.js (adicionar, editar, deletar itens) não deve ser alterada, apenas a forma como o estado inicial é carregado.
* Nenhum outro arquivo JavaScript além dos três mencionados (C-utilitarios.js, B-armazenamento-dados.js, R-modulo-configuracoes.js) deve ser modificado para esta tarefa.

Entregáveis:

Por favor, forneça o código completo e atualizado para os arquivos modificados