**src/services/LocationService.js**

```
// src/services/LocationService.js
import { ref } from 'vue';

// --- Configuração da API ---
const API_URL = import.meta.env.VITE_GOOGLE_APPS_SCRIPT_URL;

if (!API_URL) {
  console.error("VITE_GOOGLE_APPS_SCRIPT_URL não está definida no arquivo .env. As chamadas de API falharão.");
}

// Estados globais do serviço (podem ser usados pelos componentes se necessário)
const loading = ref(false);
const error = ref(null);
const successMessage = ref(null);

// Helper para limpar mensagens
function clearMessages() {
  setTimeout(() => {
    error.value = null;
    successMessage.value = null;
  }, 5000);
}

/**
 * Função interna genérica para chamar a API.
 * @param {object} params - Parâmetros para enviar (inclui 'action').
 * @param {string} method - Método HTTP ('GET', 'POST', etc.). GET é o padrão.
 * @returns {Promise<object|null>} - Os dados da resposta ou null em caso de erro.
 */
async function callApi(params = {}, method = 'GET') {
  if (!API_URL) {
    const errorMessage = "URL da API não configurada no .env.";
    error.value = errorMessage;
    console.error(errorMessage);
```

```
    return null;
  }

  loading.value = true;
  error.value = null; // Limpa erro anterior
  // Não limpa successMessage aqui, pois pode ser de uma operação anterior bem-sucedida

  let url = API_URL;
  let options = {
    method: method,
    mode: 'cors', // Necessário para requisições cross-origin
    headers: {},
  };

  const queryParams = new URLSearchParams();
  for (const key in params) {
    if (params[key] !== undefined && params[key] !== null && params[key] !== '') {
      queryParams.append(key, params[key]);
    }
  }

  if (method === 'GET' || method === 'DELETE') { // DELETE também pode usar query params
    url += `?${queryParams.toString()}`;
  } else if (method === 'POST' || method === 'PUT') {
    // Exemplo: enviando como form data (comum com Apps Script doGet/doPost simples)
    // Se o Apps Script espera JSON, use JSON.stringify e 'application/json' header
    options.headers['Content-Type'] = 'application/x-www-form-urlencoded';
    options.body = queryParams.toString();
    // Precisa adicionar a action na URL também se o doPost a ler de lá
    const actionParam = params.action ? `?action=${encodeURIComponent(params.action)}` : '';
     url += actionParam;
  }

  console.log(`Chamando API (${params.action || 'N/A'}) | Método: ${method} | URL:`, url);
  if (options.body) console.log("Body:", options.body);

  try {
```

```
    const response = await fetch(url, options);

    let result;
    const contentType = response.headers.get("content-type");

    if (contentType && contentType.includes("application/json")) {
      result = await response.json();
    } else {
      const textResponse = await response.text();
      console.warn(`Resposta da API não é JSON (Content-Type: ${contentType}). Texto:`, textResponse.substring(0, 500));
       if (response.status === 302 || textResponse.toLowerCase().includes('<title>autorização necessária</title>') || textResponse.toLowerCase().includes('required permissions')) {
           throw new Error(`Falha na API: Possível erro de permissão ou URL /dev inválida. Verifique permissões da Web App e use a URL /exec.`);
       }
       throw new Error(`Resposta inesperada (não-JSON): Status ${response.status}. Resposta: ${textResponse.substring(0, 150)}...`);
    }

    console.log("Resposta da API:", result);

    if (!response.ok || result.status === 'error' || result.error) {
      const errorMessage = result?.message || result?.error || `Erro HTTP ${response.status}`;
      throw new Error(errorMessage);
    }

    // Define mensagem de sucesso apenas para operações CRUD que não sejam de leitura
    const readActions = ['read', 'readAll', 'readLocais', 'readWaypoints', 'readFloors']; // Adicione outras ações de leitura se houver
    if (params.action && !readActions.includes(params.action)) {
        successMessage.value = result.message || 'Operação realizada com sucesso!';
        clearMessages(); // Limpa a msg de sucesso após um tempo
    }

    return result;

  } catch (err) {
```

```
    console.error(`Erro na API action=${params.action || 'N/A'}, method=${method}:`, err);
    error.value = `Falha na operação (${params.action || 'leitura'}): ${err.message}`;
    // Não limpa a msg de erro automaticamente aqui, deixa o componente decidir
    return null;
  } finally {
    loading.value = false;
  }
}

// --- Funções Exportadas ---

/**
 * Busca todos os dados iniciais necessários para o mapa.
 * @returns {Promise<{locais: Array, waypoints: Array, floors: Array}|null>}
 */
export const fetchInitialData = async () => {
  // Assume que GET sem 'action' ou com 'action=readAll' retorna tudo
  const result = await callApi({ action: 'readAll' }, 'GET'); // Ou apenas callApi()

  if (result && result.status === 'success') {
    const locais = result.locais || [];
    const waypoints = result.waypoints || [];
    const floors = result.floors || [];

    if (!Array.isArray(locais) || !Array.isArray(waypoints) || !Array.isArray(floors)) {
      console.error("Formato de dados inesperado em fetchInitialData:", result);
      error.value = "Formato de dados inválido da API.";
      return null;
    }
    console.log("Dados iniciais carregados via API.");
    return { locais, waypoints, floors };
  } else {
    // Erro já tratado em callApi e armazenado em error.value
    return null;
  }
};

```

```
/**
 * Busca apenas os locais (para o Admin Panel, por exemplo).
 * @returns {Promise<Array|null>}
 */
export const fetchAdminLocations = async () => {
    // Assume que action=read retorna apenas locais, como no LocationAdmin original
    const result = await callApi({ action: 'read' }, 'GET');
    if (result && result.status === 'success') {
        const locais = result.locais || [];
        if (!Array.isArray(locais)) {
            console.error("Formato de locais inesperado em fetchAdminLocations:", result);
            error.value = "Formato de locais inválido da API.";
            return null;
        }
        return locais;
    } else {
        return null;
    }
}


/**
 * Cria um novo local via API.
 * @param {object} locationData - { nome, endereco, andar }.
 * @returns {Promise<boolean>} - True se sucesso, false se falha.
 */
export const createLocation = async (locationData) => {
   // Usando POST como exemplo - ajuste o método se seu Apps Script usar GET
  const result = await callApi({
    action: 'create',
    nome: locationData.nome,
    endereco: locationData.endereco,
    andar: locationData.andar
  }, 'POST'); // Mude para 'GET' se necessário
  return !!result; // True se a chamada foi bem-sucedida
};

```

```
/**
 * Atualiza um local existente via API.
 * @param {object} locationData - { id, nome?, endereco?, andar? }.
 * @returns {Promise<boolean>} - True se sucesso, false se falha.
 */
export const updateLocation = async (locationData) => {
   // Usando POST como exemplo - ajuste o método se seu Apps Script usar GET
  const params = {
    action: 'update',
    id: locationData.id,
    ...(locationData.nome && { nome: locationData.nome }),
    ...(locationData.endereco && { endereco: locationData.endereco }),
    ...(locationData.andar && { andar: locationData.andar }),
    // Adicione x, y se forem editáveis
    // ...(locationData.x !== undefined && { x: locationData.x }),
    // ...(locationData.y !== undefined && { y: locationData.y }),
  };
  const result = await callApi(params, 'POST'); // Mude para 'GET' se necessário
  return !!result;
};

/**
 * Exclui um local via API.
 * @param {string|number} id - ID do local a ser excluído.
 * @returns {Promise<boolean>} - True se sucesso, false se falha.
 */
export const deleteLocation = async (id) => {
  // Usando POST como exemplo - ajuste o método se seu Apps Script usar GET
  const result = await callApi({ action: 'delete', id: id }, 'POST'); // Mude para 'GET' ou 'DELETE' se necessário
  return !!result;
};

// Exporta os estados reativos para observação externa, se necessário
export const locationServiceState = {
    loading,
    error,
    successMessage
```

```
};
```