src/services/LocationService.js

```
import { ref } from 'vue';

// Dados simulados para localizações
const locationsData = [
  { id: 1, nome: 'Bloco 1', descricao: 'Salas de aula e laboratórios', categoria: 'Acadêmico', andar: 1, latitude: -4.972138, longitude: -37.977455 },
  { id: 2, nome: 'Bloco 2', descricao: 'Departamentos e coordenações', categoria: 'Administrativo', andar: 1, latitude: -4.971956, longitude: -37.977100 },
  { id: 3, nome: 'Biblioteca', descricao: 'Acervo e salas de estudo', categoria: 'Acadêmico', andar: 1, latitude: -4.972401, longitude: -37.977012 },
  { id: 4, nome: 'Cantina', descricao: 'Área de alimentação', categoria: 'Serviços', andar: 1, latitude: -4.972587, longitude: -37.977455 },
  { id: 5, nome: 'Laboratório de Informática', descricao: 'Computadores para uso dos alunos', categoria: 'Acadêmico', andar: 2, latitude: -4.972138, longitude: -37.977256 },
  { id: 6, nome: 'Auditório', descricao: 'Eventos e palestras', categoria: 'Comum', andar: 2, latitude: -4.972300, longitude: -37.977356 },
  { id: 7, nome: 'Banheiros', descricao: 'Banheiros masculino e feminino', categoria: 'Serviços', andar: 1, latitude: -4.972456, longitude: -37.977200 },
  { id: 8, nome: 'Elevador', descricao: 'Acesso entre andares', categoria: 'Acessibilidade', andar: 1, latitude: -4.972200, longitude: -37.977300 },
  { id: 9, nome: 'Secretaria Acadêmica', descricao: 'Atendimento aos estudantes', categoria: 'Administrativo', andar: 1, latitude: -4.972050, longitude: -37.977250 },
  { id: 10, nome: 'Laboratório de Química', descricao: 'Experimentos e aulas práticas', categoria: 'Acadêmico', andar: 2, latitude: -4.972350, longitude: -37.977150 }
];

// Pontos de referência para navegação
const waypointsData = [
  { id: 1, nome: 'Entrada Principal', descricao: 'Acesso principal ao campus', latitude: -4.972700, longitude: -37.977500 },
  { id: 2, nome: 'Interseção Blocos 1-2', descricao: 'Cruzamento entre blocos', latitude: -4.972050, longitude: -37.977300 },
  { id: 3, nome: 'Rampa de Acesso', descricao: 'Acesso para cadeirantes', latitude: -4.972400, longitude: -37.977250 },
  { id: 4, nome: 'Escada Central', descricao: 'Escada para o segundo andar', latitude: -4.972200, longitude: -37.977200 },
  { id: 5, nome: 'Estacionamento A', descricao: 'Área de estacionamento frontal', latitude: -4.972800, longitude: -37.977600 }
```

```
];

// Informações sobre os andares
const floorsData = [
  { id: 1, nome: 'Térreo', descricao: 'Andar principal com recepção' },
  { id: 2, nome: 'Primeiro Andar', descricao: 'Salas administrativas e laboratórios especiais' }
];

// Estado global do serviço de localização (para monitorar estados e erros)
export const locationServiceState = {
  loading: ref(false),
  error: ref(null),
  lastUpdate: ref(null)
};

// Definindo a classe LocationService antes de usá-la
class LocationService {
  /**
   * Retorna todas as localizações
   * @returns {Promise<Array>} Lista de todas as localizações
   */
  async getAllLocations() {
    // Simulando um delay de rede
    await new Promise(resolve => setTimeout(resolve, 300));
    return [...locationsData];
  }

  /**
   * Busca uma localização pelo ID
   * @param {number} id - ID da localização
   * @returns {Promise<Object>} A localização encontrada ou undefined
   */
  async getLocationById(id) {
    await new Promise(resolve => setTimeout(resolve, 200));
    return locationsData.find(loc => loc.id === id);
  }

```

```
  /**
   * Cria uma nova localização
   * @param {Object} locationData - Dados da nova localização
   * @returns {Promise<Object>} A nova localização criada
   */
  async createLocation(locationData) {
    await new Promise(resolve => setTimeout(resolve, 400));

    // Simulando criação (na prática, isso seria feito por um backend)
    const newId = Math.max(...locationsData.map(loc => loc.id)) + 1;
    const newLocation = { id: newId, ...locationData };
    locationsData.push(newLocation);

    return newLocation;
  }

  /**
   * Atualiza uma localização existente
   * @param {number} id - ID da localização a ser atualizada
   * @param {Object} locationData - Novos dados para a localização
   * @returns {Promise<Object>} A localização atualizada
   */
  async updateLocation(id, locationData) {
    await new Promise(resolve => setTimeout(resolve, 300));

    const index = locationsData.findIndex(loc => loc.id === id);
    if (index === -1) {
      throw new Error('Localização não encontrada');
    }

    const updatedLocation = { ...locationsData[index], ...locationData };
    locationsData[index] = updatedLocation;

    return updatedLocation;
  }

  /**
```

```
   * Remove uma localização
   * @param {number} id - ID da localização a ser removida
   * @returns {Promise<boolean>} Verdadeiro se removido com sucesso
   */
  async deleteLocation(id) {
    await new Promise(resolve => setTimeout(resolve, 300));

    const index = locationsData.findIndex(loc => loc.id === id);
    if (index === -1) {
      throw new Error('Localização não encontrada');
    }

    locationsData.splice(index, 1);
    return true;
  }

  /**
   * Retorna todos os pontos de referência
   * @returns {Promise<Array>} Lista de waypoints
   */
  async getAllWaypoints() {
    await new Promise(resolve => setTimeout(resolve, 200));
    return [...waypointsData];
  }

  /**
   * Retorna informações sobre os andares
   * @returns {Promise<Array>} Lista de andares
   */
  async getFloors() {
    await new Promise(resolve => setTimeout(resolve, 100));
    return [...floorsData];
  }

  /**
   * Filtra localizações por categoria
   * @param {string} categoria - Categoria para filtrar
```

```
   * @returns {Promise<Array>} Localizações filtradas
   */
  async getLocationsByCategory(categoria) {
    await new Promise(resolve => setTimeout(resolve, 200));
    return locationsData.filter(loc => loc.categoria === categoria);
  }

  /**
   * Filtra localizações por andar
   * @param {number} andar - Número do andar
   * @returns {Promise<Array>} Localizações filtradas
   */
  async getLocationsByFloor(andar) {
    await new Promise(resolve => setTimeout(resolve, 200));
    return locationsData.filter(loc => loc.andar === andar);
  }
}

const locationService = new LocationService();

/**
 * Busca todos os dados iniciais necessários para o mapa
 * @param {boolean} forceRefresh - Se verdadeiro, recarrega os dados mesmo se já estiverem em cache
 * @returns {Promise<Object>} Objeto com locais, waypoints e andares
 */
export async function fetchInitialData(forceRefresh = false) {
  locationServiceState.loading.value = true;
  locationServiceState.error.value = null;

  try {
    // Simulando tempo de requisição
    await new Promise(resolve => setTimeout(resolve, 500));

    // Mapeia os dados para formato adequado ao mapa
    const locais = await locationService.getAllLocations();
    const waypoints = await locationService.getAllWaypoints();
    const andares = await locationService.getFloors();
```

```
172
173     // Adiciona informações adicionais para o funcionamento do mapa
174     const mappedFloors = andares.map(andar => ({
175       ...andar,
176       id: andar.id.toString(), // Garante que ID é string
177       nome: andar.nome,
178       image: `/maps/${andar.id === 1 ? 'terreo' : andar.id === 2 ? 'primeiro' : 'segundo'}.svg`
179     }));
180
181     // Mapeia locais para incluir coordenadas x, y relativas para o SVG
182     const mappedLocations = locais.map(local => ({
183       ...local,
184       andar: local.andar.toString(), // Converte andar para string para compatibilidade
185       // Valores x, y são calculados para posicionamento no mapa (hipotético)
186       x: 40 + Math.random() * 60, // Posição x aleatória entre 40% e 60% da largura do mapa
187       y: 30 + Math.random() * 50  // Posição y aleatória entre 30% e 80% da altura do mapa
188     }));
189
190     // Atualiza o timestamp de última atualização
191     locationServiceState.lastUpdate.value = new Date();
192
193     return {
194       locais: mappedLocations,
195       waypoints,
196       floors: mappedFloors
197     };
198   } catch (error) {
199     console.error('Erro ao buscar dados iniciais:', error);
200     locationServiceState.error.value = `Erro ao carregar dados: ${error.message}`;
201     return null;
202   } finally {
203     locationServiceState.loading.value = false;
204   }
205 }
206
207 export default locationService;
```