src/stores/location.js

```
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
import * as LocationService from '../services/LocationService';

export const useLocationStore = defineStore('location', () => {
  // Estado
  const locations = ref([]);
  const waypoints = ref([]);
  const floors = ref([]);
  const selectedLocal = ref(null);
  const selectedFloor = ref(null);
  const isLoading = ref(false);
  const error = ref(null);

  // Getters computados
  const currentFloorLocations = computed(() => {
    if (!selectedFloor.value) return [];
    return locations.value.filter(local => local.andar === selectedFloor.value.id);
  });

  // Ações
  async function loadInitialData(forceRefresh = false) {
    isLoading.value = true;
    error.value = null;

    try {
      const data = await LocationService.fetchInitialData(forceRefresh);

      if (data) {
        locations.value = data.locais || [];
        waypoints.value = data.waypoints || [];
        floors.value = data.floors || [];

        // Selecionar primeiro andar como padrão se não houver andar selecionado
```

```
        if (floors.value.length > 0 && !selectedFloor.value) {
          selectedFloor.value = floors.value[0];
        }

        return true;
      } else {
        error.value = 'Não foi possível carregar os dados do mapa.';
        return false;
      }
    } catch (e) {
      console.error('Erro na store ao carregar dados:', e);
      error.value = `Erro ao carregar dados: ${e.message}`;
      return false;
    } finally {
      isLoading.value = false;
    }
  }

  function selectLocal(local) {
    selectedLocal.value = local;

    // Se o local selecionado estiver em outro andar, muda para esse andar
    if (local && selectedFloor.value?.id !== local.andar) {
      const floor = floors.value.find(f => f.id === local.andar);
      if (floor) {
        selectFloor(floor);
      }
    }
  }

  function selectFloor(floor) {
    selectedFloor.value = floor;
  }

  function clearSelectedLocal() {
    selectedLocal.value = null;
  }
```

```
72
73  // Funções CRUD para administração
74  async function createLocation(localData) {
75    isLoading.value = true;
76    error.value = null;
77
78    try {
79      const success = await LocationService.createLocation(localData);
80      if (success) {
81        await loadInitialData(true);
82        return true;
83      } else {
84        error.value = 'Falha ao criar local.';
85        return false;
86      }
87    } catch (e) {
88      error.value = `Erro ao criar local: ${e.message}`;
89      return false;
90    } finally {
91      isLoading.value = false;
92    }
93  }
94
95  async function updateLocation(localData) {
96    isLoading.value = true;
97    error.value = null;
98
99    try {
100     const success = await LocationService.updateLocation(localData);
101     if (success) {
102       await loadInitialData(true);
103       return true;
104     } else {
105       error.value = 'Falha ao atualizar local.';
106       return false;
107     }
108   } catch (e) {
```

```
      error.value = `Erro ao atualizar local: ${e.message}`;
      return false;
    } finally {
      isLoading.value = false;
    }
  }

  async function deleteLocation(id) {
    isLoading.value = true;
    error.value = null;

    try {
      const success = await LocationService.deleteLocation(id);
      if (success) {
        await loadInitialData(true);
        // Se o local excluído for o selecionado, limpa a seleção
        if (selectedLocal.value?.id === id) {
          clearSelectedLocal();
        }
        return true;
      } else {
        error.value = 'Falha ao excluir local.';
        return false;
      }
    } catch (e) {
      error.value = `Erro ao excluir local: ${e.message}`;
      return false;
    } finally {
      isLoading.value = false;
    }
  }

  // Utilitários para navegação e rotas
  function findPathBetween(startPoint, endPoint) {
    // Implementação básica - conectar dois pontos em linha reta
    // Em uma implementação real, usaríamos os waypoints e algoritmo A* ou similar
    if (!startPoint || !endPoint) return [];
```

```
    return [{
      startX: startPoint.x,
      startY: startPoint.y,
      endX: endPoint.x,
      endY: endPoint.y,
      distance: Math.sqrt(
        Math.pow(endPoint.x - startPoint.x, 2) +
        Math.pow(endPoint.y - startPoint.y, 2)
      ),
      angle: Math.atan2(
        endPoint.y - startPoint.y,
        endPoint.x - startPoint.x
      ) * 180 / Math.PI
    }];
  }

  return {
    // Estado
    locations,
    waypoints,
    floors,
    selectedLocal,
    selectedFloor,
    isLoading,
    error,

    // Getters
    currentFloorLocations,

    // Ações
    loadInitialData,
    selectLocal,
    selectFloor,
    clearSelectedLocal,
    createLocation,
    updateLocation,
```

```
    deleteLocation,

    // Utilitários
    findPathBetween
  };
});
```