src/composables/useMapRouting.js

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

export function useMapRouting(userPositionRef, selectedLocationRef, currentFloorRef, mapScaleRef, mapDimensionsRef,
waypointsRef) {
  const routeSegments = ref([]);
  const debugWaypoints = ref([]);
  const routingError = ref(null);

  function calculateRouteSameFloor(start, end, andar, allWaypoints) {
    if (!start || !end || !andar || !allWaypoints) {
      console.error("Valores indefinidos em calculateRouteSameFloor");
      return [];
    }
    const waypoints = allWaypoints.filter(wp => wp.andar === andar);
    const getDist = (a, b) => Math.hypot(a.x - b.x, a.y - b.y);

    const nodes = [...waypoints];
    const tempStart = { id: 'start', x: start.x, y: start.y, neighbors: [] };
    const tempEnd = { id: 'end', x: end.x, y: end.y, neighbors: [] };

    const k = 3;
    const sortedStart = [...waypoints].sort((a, b) => getDist(a, start) - getDist(b, start));
    const sortedEnd = [...waypoints].sort((a, b) => getDist(a, end) - getDist(b, end));
    tempStart.neighbors = sortedStart.slice(0, k).map(n => n.id);
    tempEnd.neighbors = sortedEnd.slice(0, k).map(n => n.id);

    nodes.push(tempStart, tempEnd);

    const graph = {};
    nodes.forEach(wp => {
      const neighbors = wp.neighbors || wp.vizinhos || [];
      graph[wp.id] = neighbors;
    });

    const dist = {};
```

```
    const prev = {};
    const queue = new Set(nodes.map(n => n.id));

    nodes.forEach(n => dist[n.id] = Infinity);
    dist['start'] = 0;

    while (queue.size > 0) {
      const u = [...queue].reduce((minNode, node) => dist[node] < dist[minNode] ? node : minNode, [...queue][0]);
      queue.delete(u);

      const neighbors = graph[u] || [];
      neighbors.forEach(v => {
        const from = nodes.find(n => n.id === u);
        const to = nodes.find(n => n.id === v);
        if (!from || !to) return;

        const alt = dist[u] + getDist(from, to);
        if (alt < dist[v]) {
          dist[v] = alt;
          prev[v] = u;
        }
      });
    }

    const path = [];
    let u = 'end';
    while (u && u !== 'start') {
      const node = nodes.find(n => n.id === u);
      if (node) path.unshift({ x: node.x, y: node.y });
      u = prev[u];
    }

    path.unshift({ x: start.x, y: start.y });
    return path;
  }

  function calculateRouteBetweenFloors(start, end, andarStart, andarEnd, allWaypoints) {
```

```
    if (!start || !end || !andarStart || !andarEnd || !allWaypoints) {
      console.error("Valores indefinidos em calculateRouteBetweenFloors");
      return [];
    }
    if (andarStart === andarEnd) {
      return calculateRouteSameFloor(start, end, andarStart, allWaypoints);
    }

    const escadasSaida = allWaypoints.filter(wp =>
      wp.andar === andarStart && wp.tipo === 'escada' && wp.andarDestino === andarEnd
    );

    const escadasChegada = allWaypoints.filter(wp =>
      wp.andar === andarEnd && wp.tipo === 'escada' &&
      escadasSaida.some(s => s.id === wp.idLigacao)
    );

    if (!escadasSaida.length || !escadasChegada.length) {
      routingError.value = "Sem conexão entre andares.";
      return [];
    }

    const escadaInicio = escadasSaida[0];
    const escadaDestino = escadasChegada.find(e => e.idLigacao === escadaInicio.id);

    if (!escadaDestino) {
      routingError.value = "Ligação de escada não encontrada.";
      return [];
    }

    const rota1 = calculateRouteSameFloor(start, escadaInicio, andarStart, allWaypoints);
    const rota2 = [{ x: escadaDestino.x, y: escadaDestino.y }];
    const rota3 = calculateRouteSameFloor(escadaDestino, end, andarEnd, allWaypoints);

    return [...rota1, ...rota2, ...rota3];
  }

```

```
  watch(
    [userPositionRef, selectedLocationRef, currentFloorRef, mapScaleRef, mapDimensionsRef, waypointsRef],
    () => {
      try {
        if (!userPositionRef.value || !selectedLocationRef.value || !currentFloorRef.value || !waypointsRef.value) {
          routeSegments.value = [];
          return;
        }

        const start = userPositionRef.value;
        const endLocation = selectedLocationRef.value;
        const andarAtual = currentFloorRef.value;
        const waypoints = waypointsRef.value || [];

        routingError.value = null;

        if (!start || !endLocation) {
          routeSegments.value = [];
          return;
        }

        const end = { x: endLocation.x, y: endLocation.y };
        const andarDestino = endLocation.andar;

        const path = calculateRouteBetweenFloors(start, end, andarAtual, andarDestino, waypoints);

        debugWaypoints.value = waypoints;

        routeSegments.value = [];

        for (let i = 0; i < path.length - 1; i++) {
          const p1 = path[i];
          const p2 = path[i + 1];

          const dx = p2.x - p1.x;
          const dy = p2.y - p1.y;
          const length = Math.sqrt(dx * dx + dy * dy);
```

```
          const angle = Math.atan2(dy, dx) * (180 / Math.PI);

          routeSegments.value.push({
            x: p1.x,
            y: p1.y,
            length,
            angle
          });
        }
      } catch (error) {
        console.error("Erro durante o calculo da rota:", error);
        routingError.value = "Erro interno ao calcular a rota.";
        routeSegments.value = [];
      }
    },
    { immediate: true }
  );

  return {
    routeSegments,
    debugWaypoints,
    routingError
  };
}
```