

```

1 <template>
2   <div class="app-container">
3     <button class="menu-toggle-button" @click="toggleMenu" title="Abrir/Fechar Menu">≡</button>
4
5     <AppSidebar
6       :is-open="isMenuOpen"
7       :all-locais="allLocais"
8       :floors="floors"
9       :current-floor-id="currentFloorId"
10      :selected-local-id="selectedLocal?.id"
11      :loading="loading.initialData" :error="error.locais" @update:currentFloorId="changeFloor"
12      @select-local="handleLocalSelect"
13      @close="closeMenu" />
14
15     <div class="main-content" style="flex-grow: 1; position: relative;">
16       <MapDisplay
17         :map-image-url="currentFloorMap"
18         :locais-on-floor="locaisAtualAndar"
19         :user-position="userPosition"
20         :route-segments="routeSegments"
21         :debug-waypoints="debugWaypoints" :selected-local-id="selectedLocal?.id"
22         :popup-info="popupInfo"
23         :map-scale="mapScale"
24         :loading="loading.map" @map-click="handleMapClick"
25         @marker-click="handleLocalSelect"
26         @marker-mouseenter="showPopup"
27         @marker-mouseleave="hidePopup"
28         @update:map-dimensions="mapDimensions = $event" />
29
30       <MapControls
31         @zoom-in="zoomIn"
32         @zoom-out="zoomOut"
33         @update-location="triggerAutoLocation"
34         @set-location-manual="triggerManualLocationMode" />
35
36     <div v-if="overallLoading && !displayError" class="loading">

```

```

37         <span v-if="loading.initialData">Carregando dados do mapa...</span>
38         <span v-else-if="isGettingLocation">Obtendo localização...</span>
39         <span v-else>Carregando...</span>
40     </div>
41
42
43     <div v-if="displayError" class="error">{{ displayError }}</div>
44
45     <div v-if="settingUserLocationMode" class="manual-location-prompt">
46         {{ manualModeMessage }}
47     </div>
48
49     <router-view />
50 </div>
51 </div>
52 </template>
53
54 <script setup>
55 import { ref, computed, onMounted, watch } from 'vue';
56 import AppSidebar from '@components/AppSidebar.vue';
57 import MapDisplay from '@components/MapDisplay.vue';
58 import MapControls from '@components/MapControls.vue';
59 import { useGeolocation } from '@composables/useGeolocation.js';
60 import { useMapInteraction } from '@composables/useMapInteraction.js';
61 import { useMapRouting } from '@composables/useMapRouting.js';
62 // Importa a função unificada do serviço
63 import { fetchInitialData, locationServiceState } from '@services/LocationService.js';
64
65 // --- Estado Principal ---
66 const allLocais = ref([]);
67 const floors = ref([]);
68 const waypoints = ref([]); // Novo ref para waypoints
69 const selectedLocal = ref(null);
70 const currentFloorId = ref('terreo'); // Andar inicial padrão
71 const isMenuOpen = ref(false);
72 const mapDimensions = ref({ width: 0, height: 0 });
73

```

```
74 // Estados de Loading e Erro - mais detalhados
75 const loading = ref({
76   initialData: true, // Combina loading de locais, waypoints, andares
77   map: true, // Pode ser redundante se o mapa só carrega após initialData
78 });
79 const error = ref({
80   initialData: null, // Erro ao carregar dados gerais
81   locais: null, // Mantido para erro específico da sidebar, se necessário
82   routing: null, // Vindo do useMapRouting
83   geolocation: null // Vindo do useGeolocation
84 });
85 const manualModeMessage = ref(''); // Mensagem para modo manual
86
87 // --- Composables ---
88 const {
89   userPosition,
90   geolocationError,
91   isGettingLocation,
92   getUserLocationAuto,
93   setUserLocationManually
94 } = useGeolocation();
95
96 const {
97   mapScale,
98   popupInfo,
99   settingUserLocationMode,
100   zoomIn,
101   zoomOut,
102   showPopup,
103   hidePopup,
104   handleMapClick: handleMapClickInteraction,
105   enableSetUserLocationMode,
106 } = useMapInteraction();
107
108 // Passa o ref de waypoints para o composável de roteamento
109 const {
110   routeSegments,
```

```

111     hasRoute,
112     debugWaypoints, // Recebe waypoints de debug
113     routingError: routeCalcError,
114     calculateRoute // Pode não ser mais necessário chamar manualmente
115 } = useMapRouting(userPosition, selectedLocal, currentFloorId, mapScale, mapDimensions, waypoints);
116
117 // --- Dados Computados ---
118 const currentFloorData = computed(( ) => floors.value.find(f => f.id === currentFloorId.value));
119 const currentFloorMap = computed(( ) => currentFloorData.value?.image || '');
120 const locaisAtualAndar = computed(( ) => allLocais.value.filter(local => local.andar === currentFloorId.value));
121 const overallLoading = computed(( ) => loading.value.initialData || isGettingLocation.value);
122 // Combina erros para exibição principal (pode priorizar)
123 const displayError = computed(( ) =>
124     error.value.initialData ||
125     error.value.geolocation ||
126     error.value.routing ||
127     locationServiceState.error.value // Observa erro global do serviço tbm
128 );
129
130
131 // --- Métodos / Manipuladores de Eventos ---
132 const toggleMenu = () => { isMenuOpen.value = !isMenuOpen.value; };
133 const closeMenu = () => { isMenuOpen.value = false; };
134
135 const loadInitialData = async () => {
136     loading.value.initialData = true;
137     loading.value.map = true; // Assume que mapa depende dos dados
138     error.value.initialData = null;
139     error.value.locais = null; // Limpa erro específico também
140     try {
141         const initialData = await fetchInitialData(); // Chama a função unificada
142         if (initialData) {
143             allLocais.value = initialData.locais;
144             floors.value = initialData.floors;
145             waypoints.value = initialData.waypoints; // Atualiza o ref de waypoints
146
147             // Define andar padrão se necessário (após carregar andares)

```

```

148     if (floors.value.length > 0 && !floors.value.some(f => f.id === currentFloorId.value)) {
149         currentFloorId.value = floors.value[0].id;
150     }
151     // Verifica se há locais, andares ou waypoints
152     if (allLocais.value.length === 0) console.warn("Nenhum local carregado da API.");
153     if (floors.value.length === 0) console.warn("Nenhum andar carregado da API.");
154     if (waypoints.value.length === 0) console.warn("Nenhum waypoint carregado da API.");
155
156
157     } else {
158         // Erro já deve estar em locationServiceState.error ou error.value do serviço
159         error.value.initialData = locationServiceState.error.value || "Falha ao carregar dados iniciais.";
160     }
161 } catch (err) {
162     // Captura erros inesperados no processo
163     console.error("Erro crítico ao carregar dados iniciais:", err);
164     error.value.initialData = `Erro inesperado: ${err.message}`;
165 } finally {
166     loading.value.initialData = false;
167     loading.value.map = false; // Permite que o mapa seja exibido
168 }
169 };
170
171 const changeFloor = (newFloorId) => {
172     if (currentFloorId.value !== newFloorId) {
173         console.log("Mudando para andar:", newFloorId);
174         currentFloorId.value = newFloorId;
175         selectedLocal.value = null; // Deseleciona local ao mudar de andar
176         hidePopup(); // Esconde popup
177         // A rota será recalculada automaticamente pelo watcher em useMapRouting
178     }
179 };
180
181
182 const handleLocalSelect = (local) => {
183     if (!local) return;
184     selectedLocal.value = local;

```

```

185 // Se o local selecionado estiver em outro andar, muda para ele
186 if (local.andar !== currentFloorId.value) {
187     changeFloor(local.andar); // Usa a função changeFloor
188 }
189 // Fecha o menu ao selecionar um local
190 closeMenu();
191 };
192
193 // Função chamada pelo MapDisplay quando o mapa é clicado
194 const handleMapClick = (event, mapRect) => {
195     // Passa para o composable de interação, fornecendo callbacks
196     handleMapClickInteraction(
197         event,
198         mapRect,
199         (xPercent, yPercent) => { // Callback para definir localização manual
200             setUserLocationManually (xPercent, yPercent);
201             error.value.geolocation = null; // Limpa msg de erro de geo
202             manualModeMessage.value = ''; // Limpa msg de modo manual
203         },
204         () => { // Callback para tentar fechar o menu
205             // Fecha o menu somente se estiver aberto e o clique não foi no botão toggle
206             if (isMenuOpen.value && !event.target.closest('.menu-toggle-button')) {
207                 closeMenu();
208             }
209         }
210     );
211
212     // Se NÃO estiver no modo de definir localização, um clique no mapa deseleciona o local?
213     if (!settingUserLocationMode.value) {
214         selectedLocal.value = null;
215         hidePopup();
216     }
217 };
218
219 // Inicia a busca automática de localização (acionado pelo botão)
220 const triggerAutoLocation = () => {
221     getUserLocationAuto(); // Chama a função do composable

```

```
222     closeMenu(); // Fecha o menu ao tentar obter localização
223 };
224
225 // Ativa o modo de definição manual (acionado pelo botão)
226 const triggerManualLocationMode = () => {
227     enableSetUserLocationMode(); // Ativa o modo no composabile de interação
228     manualModeMessage.value = "Clique no mapa para definir sua localização."; // Informa o usuário
229     error.value.geolocation = null; // Limpa outros erros de geo
230     closeMenu();
231 };
232
233 // --- Watchers ---
234
235 // Observa erros dos composables para atualizar o estado de erro local
236 watch(geolocationError, (newError) => {
237     // Só atualiza o erro de geolocalização se NÃO estiver no modo de definição manual
238     if (!settingUserLocationMode.value) {
239         error.value.geolocation = newError;
240         manualModeMessage.value = ''; // Garante que a msg manual seja limpa se houver erro real
241     }
242 });
243 watch(routeCalcError, (newError) => { error.value.routing = newError; });
244
245 // Observa mudança no andar (já tratado em changeFloor, mas pode ter lógica adicional aqui se necessário)
246 // watch(currentFloorId, (newFloor, oldFloor) => { ... });
247
248
249 // Limpa a seleção e o popup se o local selecionado não pertencer mais ao andar atual
250 // (Embora changeFloor já deselectione, este watcher pode ser útil se currentFloorId mudar por outros meios)
251 watch(selectedLocal, (newLocal) => {
252     if (newLocal && newLocal.andar !== currentFloorId.value) {
253         // Isso não deveria acontecer se changeFloor deselectiona, mas como segurança:
254         selectedLocal.value = null;
255         hidePopup();
256     }
257 });
258
```

```
259
260 // Observa o estado de erro global do LocationService
261 watch(() => locationServiceState.error.value, (newServiceError) => {
262     if (newServiceError && !error.value.initialData) {
263         // Mostra erro do serviço se não houver erro mais específico já sendo mostrado
264         // Poderia ser mais granular
265         console.error("Erro detectado no LocationService:", newServiceError);
266     }
267 });
268
269
270 // --- Ciclo de Vida ---
271 onMounted(() => {
272     loadInitialData(); // Carrega locais, waypoints, andares da API
273     getUserLocationAuto(); // Tenta obter localização ao iniciar
274 });
275
276 </script>
277
278 <style scoped>
279 /* Estilos específicos para o layout, se necessário */
280 .main-content {
281     width: 100%;
282     height: 100vh;
283 }
284
285 /* Ajuste para sobrepor o erro/loading global se necessário */
286 .error, .loading {
287     z-index: 1100; /* Acima de outros elementos */
288 }
289
290 /* Estilo para a mensagem do modo manual */
291 .manual-location-prompt {
292     position: absolute;
293     top: 10%;
294     left: 50%;
295     transform: translateX(-50%);
```



```
296     background: lightblue;
297     color: black;
298     border: 1px solid blue;
299     padding: 10px 15px;
300     border-radius: 5px;
301     z-index: 1100;
302     box-shadow: 0 2px 5px rgba(0,0,0,0.2);
303     font-weight: bold;
304     text-align: center;
305 }
306
307 /* Estilos para o botão de toggle (pode estar em main.css) */
308 .menu-toggle-button {
309     position: absolute;
310     top: 15px;
311     left: 15px;
312     z-index: 1001; /* Acima de tudo */
313     background-color: var(--button-bg-color, #f0f0f0);
314     color: var(--text-color, #333);
315     border: 1px solid var(--border-color, #ccc);
316     border-radius: 4px;
317     padding: 8px 12px;
318     font-size: 1.5em;
319     cursor: pointer;
320     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
321     transition: background-color 0.2s ease;
322 }
323 .menu-toggle-button:hover {
324     background-color: var(--button-hover-bg-color, #e0e0e0);
325 }
326
327 /* Container principal (pode estar em main.css) */
328 .app-container {
329     position: relative;
330     width: 100vw;
331     height: 100vh;
332     display: flex;
```

```
333 |     overflow: hidden;
334 | }
335 | </style>
```