You're right! Let me simplify this with a single component approach that handles dynamic nested menus. This will be much cleaner and easier to maintain.

Single Component Solution

```typescript
// nested-menu.component.ts
import { Component, Input, Output, EventEmitter, ViewChild, TemplateRef } from
'@angular/core';
import { MatMenu, MatMenuTrigger } from '@angular/material/menu';

export interface MenuItem {
  id: string;
  label: string;
  icon?: string;
  route?: string;
  children?: MenuItem[];
  data?: any;
}

@Component({
  selector: 'app-nested-menu',
  templateUrl: './nested-menu.component.html',
  styleUrls: ['./nested-menu.component.css']
})
export class NestedMenuComponent {
  @Input() menuItems: MenuItem[] = [];
  @Input() level: number = 0;
  @Output() itemSelected = new EventEmitter<MenuItem>();

  @ViewChild(MatMenuTrigger, { static: true }) menuTrigger!: MatMenuTrigger;

  // Track open submenus to prevent closing when hovering between items
  openSubmenus: Set<string> = new Set();

  handleItemClick(item: MenuItem): void {
    if (!item.children || item.children.length === 0) {
      this.itemSelected.emit(item);
      this.menuTrigger.closeMenu();
    }
  }

  shouldShowSubmenu(item: MenuItem): boolean {
    return !!item.children && item.children.length > 0;
```

```
  }

  onSubmenuOpened(itemId: string): void {
    this.openSubmenus.add(itemId);
  }

  onSubmenuClosed(itemId: string): void {
    this.openSubmenus.delete(itemId);
  }

  isSubmenuOpen(itemId: string): boolean {
    return this.openSubmenus.has(itemId);
  }
}
```

```html
<!-- nested-menu.component.html -->
<mat-menu #menu="matMenu" class="nested-menu">
  <ng-template matMenuContent>
    <ng-container *ngFor="let item of menuItems">
      <!-- Menu item with children (has submenu) -->
      <button *ngIf="shouldShowSubmenu(item)"
              mat-menu-item
              [matMenuTriggerFor]="submenu"
              [matMenuTriggerData]="{ items: item.children }"
              (mouseenter)="onSubmenuOpened(item.id)"
              (mouseleave)="onSubmenuClosed(item.id)"
              (click)="$event.stopPropagation()">

        <mat-icon *ngIf="item.icon">{{ item.icon }}</mat-icon>
        <span>{{ item.label }}</span>
        <mat-icon class="submenu-arrow">arrow_right</mat-icon>
      </button>

      <!-- Leaf menu item (no children) -->
      <button *ngIf="!shouldShowSubmenu(item)"
              mat-menu-item
              (click)="handleItemClick(item)">

        <mat-icon *ngIf="item.icon">{{ item.icon }}</mat-icon>
        <span>{{ item.label }}</span>
      </button>
    </ng-container>
```

```
  </ng-template>
</mat-menu>

<!-- Recursive submenu template -->
<ng-template #submenu let-items="items">
  <app-nested-menu [menuItems]="items"
                   [level]="level + 1"
                   (itemSelected)="itemSelected.emit($event)">
  </app-nested-menu>
</ng-template>
```

```css
/* nested-menu.component.css */
.nested-menu {
  max-height: 80vh;
  overflow-y: auto;
}

.submenu-arrow {
  margin-left: auto;
  font-size: 18px;
  height: 18px;
  width: 18px;
}

/* Style for different nesting levels */
.menu-level-1 { margin-left: 8px; }
.menu-level-2 { margin-left: 16px; }
.menu-level-3 { margin-left: 24px; }
```

Parent Component Usage

```typescript
// parent.component.ts
import { Component } from '@angular/core';
import { MenuItem } from './nested-menu.component';

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
```

```
export class ParentComponent {
  menuItems: MenuItem[] = [
    {
      id: 'home',
      label: 'Home',
      icon: 'home',
      route: '/home'
    },
    {
      id: 'products',
      label: 'Products',
      icon: 'shopping_cart',
      children: [
        {
          id: 'electronics',
          label: 'Electronics',
          children: [
            { id: 'phones', label: 'Phones', route: '/products/phones' },
            { id: 'laptops', label: 'Laptops', route: '/products/laptops' },
            { id: 'tablets', label: 'Tablets', route: '/products/tablets' }
          ]
        },
        {
          id: 'clothing',
          label: 'Clothing',
          children: [
            {
              id: 'men',
              label: 'Men',
              children: [
                { id: 'shirts', label: 'Shirts', route: '/products/men/shirts' },
                { id: 'pants', label: 'Pants', route: '/products/men/pants' }
              ]
            },
            {
              id: 'women',
              label: 'Women',
              children: [
                { id: 'dresses', label: 'Dresses', route: '/products/women/dresses' },
                { id: 'skirts', label: 'Skirts', route: '/products/women/skirts' }
              ]
            }
          ]
        }
```

```
      ]
    },
    {
      id: 'services',
      label: 'Services',
      icon: 'build',
      children: [
        { id: 'consulting', label: 'Consulting', route: '/services/consulting' },
        { id: 'support', label: 'Support', route: '/services/support' },
        { id: 'training', label: 'Training', route: '/services/training' }
      ]
    },
    {
      id: 'about',
      label: 'About Us',
      icon: 'info',
      route: '/about'
    }
  ];

  onMenuItemSelected(item: MenuItem): void {
    console.log('Menu item selected:', item);
    if (item.route) {
      // Navigate to the route
      // this.router.navigate([item.route]);
    }
    // Handle other menu item actions
  }
}
```

```html
<!-- parent.component.html -->
<mat-toolbar color="primary">
  <span>My Application</span>

  <nav class="toolbar-nav">
    <button *ngFor="let item of menuItems"
            mat-button
            [matMenuTriggerFor]="item.children ? mainMenu : null"
            [matMenuTriggerData]="{ items: item.children }"
            (click)="!item.children && onMenuItemSelected(item)">

      <mat-icon *ngIf="item.icon">{{ item.icon }}</mat-icon>
```

```
      {{ item.label }}
      <mat-icon *ngIf="item.children">arrow_drop_down</mat-icon>
    </button>
  </nav>
</mat-toolbar>

<main>
  <router-outlet></router-outlet>
</main>

<!-- Main menu template -->
<ng-template #mainMenu let-items="items">
  <app-nested-menu [menuItems]="items"
                   (itemSelected)="onMenuItemSelected($event)">
  </app-nested-menu>
</ng-template>
```

```css
/* parent.component.css */
.toolbar-nav {
  margin-left: auto;
  display: flex;
  gap: 8px;
}

.toolbar-nav button {
  display: flex;
  align-items: center;
  gap: 4px;
}
```

Module Setup

```typescript
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { RouterModule } from '@angular/router';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatButtonModule } from '@angular/material/button';
import { MatMenuModule } from '@angular/material/menu';
```

```
import { MatIconModule } from '@angular/material/icon';

import { AppComponent } from './app.component';
import { ParentComponent } from './parent.component';
import { NestedMenuComponent } from './nested-menu.component';

@NgModule({
  declarations: [
    AppComponent,
    ParentComponent,
    NestedMenuComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    RouterModule.forRoot([]),
    MatToolbarModule,
    MatButtonModule,
    MatMenuModule,
    MatIconModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Key Features of This Approach:

1. Single Component: Only one component handles all nesting levels
2. Recursive Template: Uses ng-template with matMenuTriggerData for recursion
3. Simple Data Flow: Clean input/output structure
4. No Complex Service Needed: Eliminates the need for menu instance management
5. Automatic Level Handling: The component automatically handles any nesting depth
6. Hover Support: Proper submenu hover behavior
7. Clean Separation: Parent component only provides data and handles actions

Benefits:

· Simplified Code: No complex ViewChild/ViewChildren management
· No Timing Issues: No ngAfterViewInit problems
· Easy Maintenance: Only one component to maintain
· Better Performance: No hidden DOM elements for submenus
· Clean Architecture: Proper separation of concerns

This approach is much simpler and follows Angular Material's intended usage patterns. The recursive template approach is the recommended way to handle nested menus in Angular Material.