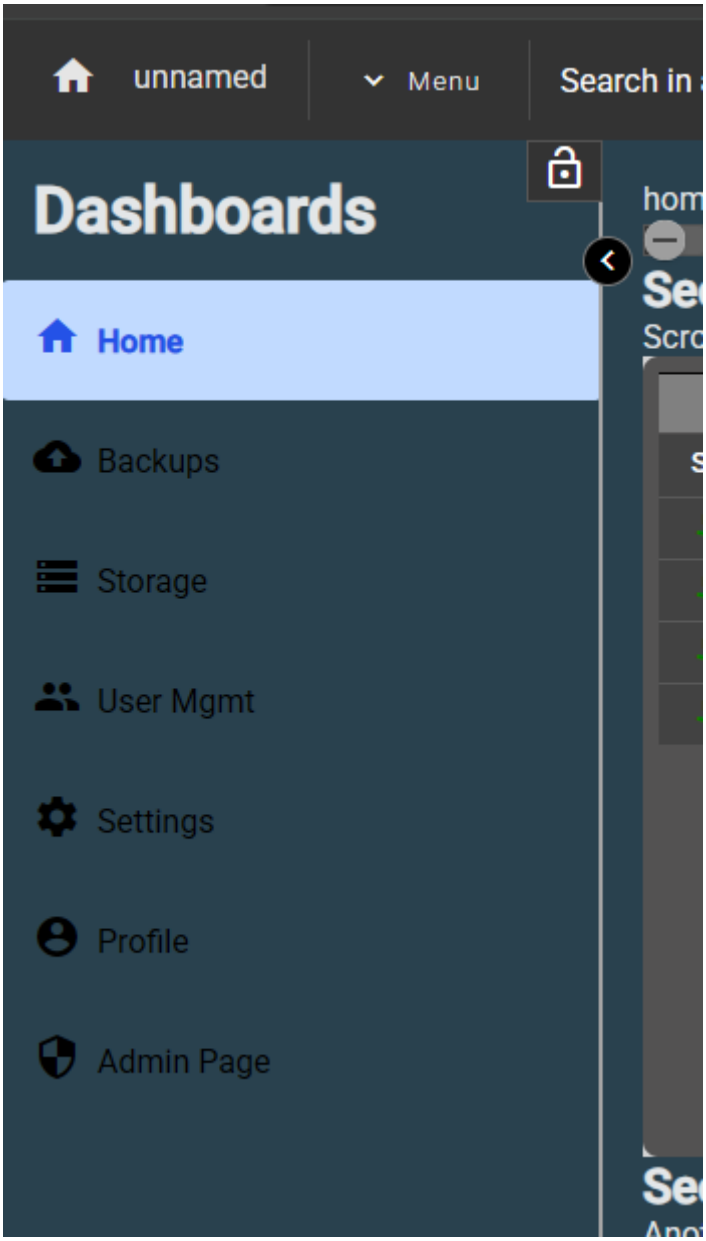
Swapping Side-Nav Groups

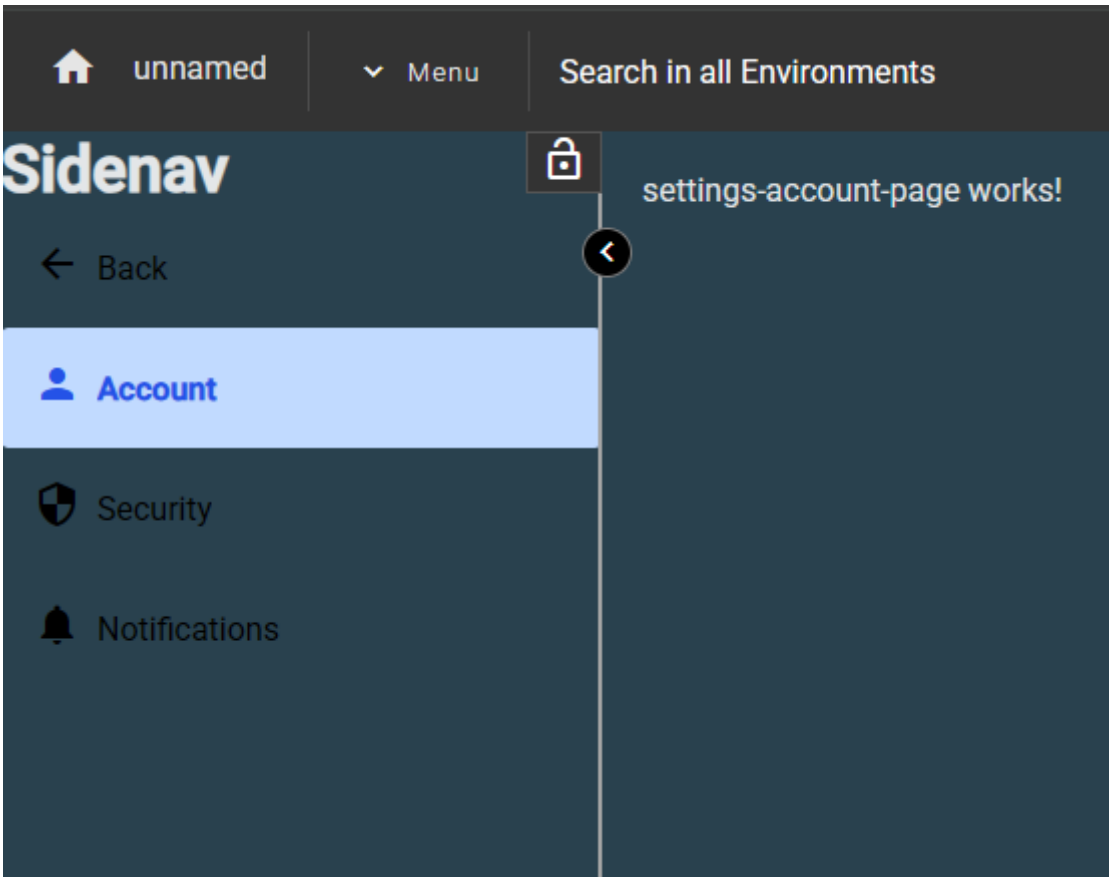
As your application grows in complexity, it will be come necessary to add layers to your side navigation menus.

One way to do this, is to swap in the different layers, based on context.

For example: Below is a side-nav menu with a top-level entry list:



If the user presses "Settings", a secondary list is swapped in, like this:



The user can navigate the settings related pages.

And, they can go back to the top-level list by pressing 'Back'.

The side nav service does the dirty work of tracking a stack of menus as they are drilled down and drilled up (pushed and popped).

All the consuming logic has to do is, push() a new menu, to 'drill down', and pop() the menu when 'drilling up'.

Library Usage

Containing Component Needs

Whatever you define that will hold the side nav element (Likely an app-layout component), will need to:

- Push the initial menu content to the side nav service, when opening.
- Pop the menu set, when closing.

Here's what a partial layout component would need, to do the above:

```
import { Component, ChangeDetectorRef } from '@angular/core';
import { AfterViewInit, OnDestroy } from '@angular/core';

import { DefaultSidenavComponent } from 'somewhere';

export class AppLayoutComponent implements AfterViewInit, OnDestroy {

  constructor(public sidenavService: SidenavService,
              private cdr: ChangeDetectorRef)
  {
  }

  ngAfterViewInit() {
    this.sidenavService.push(DefaultSidenavComponent);
    this.sidenavService.CollapseSidenav();
    this.cdr.detectChanges();
  }

  ngOnDestroy() {
    this.sidenavService.pop();
    this.cdr.detectChanges();
  }
```

Routing Side Nav Service

To automate menu content changes, your app can include a routing side nav service, that will hook into the routing flow, to watch for navigation changes that require a different menu list.

Here's a working routing side nav service that handles two menu sets: default and settings.

```
import { Injectable, OnDestroy } from '@angular/core';

import { Router } from '@angular/router';
import { NavigationEnd } from '@angular/router';

import { filter, map } from 'rxjs';

// Need a reference to the side nav service...
import { SidenavService } from 'side-nav';
```

```
// Import the side nav entry components...
import { SettingsSidenavComponent } from './settings-sidenav/settings-sidenav.component';
import { DefaultSidenavComponent } from './default-sidenav/default-sidenav.component';



@Injectable({
  providedIn: 'root'
})
export class RoutingSidenavService implements OnDestroy
{
  //#region Private Fields

  static lastusedInstanceId: number = 0;
  private instanceId: number = 0;

  // Store the last known side-nav key that is being presented...
  private currentSidenavKey: string | null = null;

  //#endregion



  //#region ctor / dtor

  constructor(private _snsvc:SidenavService,
              private router: Router)
  {
    RoutingSidenavService.lastusedInstanceId++;
    this.instanceId = RoutingSidenavService.lastusedInstanceId;
    console.log("RoutingSidenavService-" + this.instanceId + ":constructor - triggered.");
    // Subscribe to router events...
    this.router.events
      .pipe(
        // Only respond to completed navigation events (i.e., when a route is fully activated)...
        filter(event => event instanceof NavigationEnd),
        // Start at the root route...
        map(() => this.router.routerState.root),
        // Traverse down to the deepest active route (where the sidenav data is most likely defined)...
        map(route => {
          while (route.firstChild) route = route.firstChild;
          return route;
```

```
      }),
      // Extract the `sidenav` metadata from the route's snapshot (defined in the route config `data`)...
      map(route => route.snapshot.data['sidenav']),
      // Filter for only events, where the sidenav key is different than what we have...
      // This prevents swapping in side-nav entries when the key hasn't changed.
      filter(sidenavKey => sidenavKey !== this.currentSidenavKey)
    )
    // This is our callback logic that will make the request to swap in the needed side nav entries,
    //  based on the side-nav key.
    .subscribe((sidenavKey) => {
      // Store the new side nav key...
      this.currentSidenavKey = sidenavKey;

      // Perform the necessary swap...
      switch (sidenavKey) {
        case 'settings':
          this._snsvc.push(SettingsSidenavComponent);
          break;
        case 'default':
        default:
          this._snsvc.pop();
          break;
      }
    });
  }

  ngOnDestroy()
  {
    console.log("RoutingSidenavService-" + this.instanceId + ":ngOnDestroy - triggered.");
    // Drill up as far as we can nest...
    this._snsvc.pop();
    this._snsvc.pop();
  }

  //#endregion
}
```

Menu Content Implementations

And, you need to generate the menu entry content for a default menu, and any other menu sets you need.

Default Menu Content

Here's what a default menu looks like:

```
<app-sidenav-link routerLink="/home">
  <mat-icon icon>home</mat-icon>
  Home
</app-sidenav-link>

<app-sidenav-link routerLink="/backups">
  <mat-icon icon>backup</mat-icon>
  Backups
</app-sidenav-link>

<app-sidenav-link routerLink="/storage">
  <mat-icon icon>storage</mat-icon>
  Storage
</app-sidenav-link>

<app-sidenav-link routerLink="/users">
  <mat-icon icon>people</mat-icon>
  User Mgmt
</app-sidenav-link>



<app-sidenav-link routerLink="/settings">
  <mat-icon icon>settings</mat-icon>
  Settings
</app-sidenav-link>

<app-sidenav-link routerLink="/profile">
  <mat-icon icon>account_circle</mat-icon>
  Profile
</app-sidenav-link>

<app-sidenav-link routerLink="/admin">
  <mat-icon icon>security</mat-icon>
  Admin Page
```

```
</app-sidenav-link>
```

And, it's component definition:

```
import { Component } from '@angular/core';

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

import { SidenavLinkComponent } from 'side-nav';



@Component({
  // selector: 'default-sidenav',
  imports:
  [
    MatIconModule,
    SidenavLinkComponent
  ],
  templateUrl: './default-sidenav.component.html',
  styleUrl: './default-sidenav.component.scss'
})
export class DefaultSidenavComponent {

}
```

And, here's what a secondary menu set looks like.
Our example is a second-level Settings menu.

```
<app-sidenav-link
  routerLink="/"
  [routerLinkActiveOptions]="{ exact: true }">
  <mat-icon icon> arrow_back </mat-icon>
  Back
</app-sidenav-link>

<app-sidenav-link routerLink="/settings">
  <mat-icon icon> settings </mat-icon>
  Settings Home
</app-sidenav-link>
```

```
<app-sidenav-link routerLink="/settings/account">
  <mat-icon icon> person </mat-icon>
  Account
</app-sidenav-link>

<app-sidenav-link routerLink="/settings/security">
  <mat-icon icon> security </mat-icon>
  Security
</app-sidenav-link>

<app-sidenav-link routerLink="/settings/notifications">
  <mat-icon icon> notifications </mat-icon>
  Notifications
</app-sidenav-link>
```

And, its corresponding component definition:

```
import { Component } from '@angular/core';

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

import { SidenavLinkComponent } from 'side-nav';



@Component({
  // selector: 'settings-sidenav',
  imports:
  [
    MatIconModule,
    SidenavLinkComponent
  ],
  templateUrl: './settings-sidenav.component.html',
  styleUrl: './settings-sidenav.component.scss'
})
export class SettingsSidenavComponent {

}
```

Conclusion

With the above elements defined in your app, you should have a working side nav implementation.

Revision #2
Created 24 March 2025 12:35:05 by glwhite
Updated 26 March 2025 01:52:42 by glwhite