Angular Material Dark Mode Theming

Here are the steps required to implement light/dark mode in Angular 17 with Angular Material.

Once done, your app will be able to toggle between light and dark modes.

Locked Theme

If you want your app to define a theme, and prevent user changes, set the theme name in the environment.ts to one of these:

- lock-light
- lock-dark

Styles.scss

Update the styles.scss file to add this at the top:

```
/* Start of light-dark mode styling */
@use 'assets/themes.scss' as themes;
/* End of light-dark modestyling */
```

Themes.scss

Create a themes.scss file in your app's assets folder.

Give it this content:

```
/* This file includes light/dark mode theming for Angular Material and non-Material components.
    It is referenced by an include in styles.scss.

    It includes some custom coloring for Material card and dialog components.
    And, it includes custom coloring for non-Material elements.
*/

@use '@angular/material' as mat;

```

```scss
@include mat.core();

// Define the default (light) theme
$light-theme: mat.define-light-theme((
  color: (
    primary: mat.define-palette(mat.$indigo-palette),
    accent: mat.define-palette(mat.$pink-palette),
  ),
  typography: mat.define-typography-config(),
  density: 0
));

// Define the dark theme
$dark-theme: mat.define-dark-theme((
  color:
  (
    primary: mat.define-palette(mat.$blue-gray-palette),
    accent: mat.define-palette(mat.$deep-orange-palette),
    background:
    (
      background: #29414e, // Custom dark background color
      card: #252525,       // Background for Material cards
      dialog: #2A2A2A,     // Background for dialogs/modals
      hover: #333333,      // Background when hovering Material elements
    )
  )
));

// Apply the light theme globally by default
@include mat.all-component-themes($light-theme);

// Manually set background colors
body {
  background-color: #ffffff; // Light theme background
  color: rgba(0, 0, 0, 0.87); // Light theme text color
  transition: background 0.3s ease-in-out, color 0.3s ease-in-out;
}

// Dark theme styles
.dark-theme {
```

```
  @include mat.all-component-colors($dark-theme);

  // #29414e
  background-color: #29414e; // Dark theme background
  color: rgba(255, 255, 255, 0.87); // Dark theme text color
}

/* The following will assign light/dark mode theme colors to non-Material elements */

// Style additional elements
.dark-theme h1,
.dark-theme h2,
.dark-theme h3,
.dark-theme p,
.dark-theme a {
  color: rgba(255, 255, 255, 0.87); // Adjust text color for dark mode
}

.dark-theme a:hover {
  color: #bb86fc; // Optional: Change hover color for dark mode
}

.dark-theme button {
  background-color: #333; // Dark theme for non-Material buttons
  color: white;
  border: 1px solid #555;
}

.dark-theme input {
  background-color: #222; // Dark theme for input fields
  color: white;
  border: 1px solid #555;
}
```

Color Theme Service

We need a service to centrally manage color theming.

Here's what it looks like:

```
/*
  This service provides a central point for color theme changes to the app.
  Callers can reference this service, and call setTheme() or toggleTheme().

*/

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

import { BehaviorSubject } from 'rxjs';

import { IEnvironment, ENVIRONMENT_INJTOKEN } from 'lib-oga-webui-sharedkernel';


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

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

  private isDarkThemeSubject = new BehaviorSubject<boolean>(false);
  isDarkTheme$ = this.isDarkThemeSubject.asObservable();

  public lockedtheme: boolean = false;

  //#endregion


  //#region ctor / dtor

  constructor(
            // Injecting app-level configuration for use...
            // See this:
https://oga.atlassian.net/wiki/spaces/~311198967/pages/131760134/Angular+Expose+Environment+Config+to
+DI
            @Inject(ENVIRONMENT_INJTOKEN) private env: IEnvironment)
  {
```

```
  ColorThemingService.lastusedInstanceId++;
  this.instanceId = ColorThemingService.lastusedInstanceId;
  console.log("ColorThemingService-" + this.instanceId + ":constructor - triggered.");

  // Automatically load the previous theme when the service is created...
  this._loadTheme();

  console.log("ColorThemingService-" + this.instanceId + ":constructor - ended.");
}

ngOnDestroy()
{
  console.log("ColorThemingService-" + this.instanceId + ":ngOnDestroy - triggered.");
}

//#endregion


//#region Private Methods

// Private method used to initialize the theme, on startup, from stored preferences or platform defaults.
private _loadTheme()
{
  console.log("ColorThemingService-" + this.instanceId + ":_loadTheme - triggered.");

  // Attempt to load theme from localStorage...
  const storedTheme = localStorage.getItem('theme');

  // Attempt to get the platform preference...
  const prefersDark = window.matchMedia('(prefers-color-scheme: dark)').matches;

  // And, get the default theme from the environment...
  const envtheme = this.env.default_ThemeName;

  // Check if the environment allows user theme choice...
  if(envtheme === 'light')
  {
    // The environment specifies a light theme.

    // Lock the theme...
```

```
    this.lockedtheme = true;

    // Set the light theme...
    this.setTheme(false);
  }
  else if(envtheme === 'dark')
  {
    // The environment specifies a dark theme.

    // Set the dark theme...
    this.setTheme(true);

    // Lock the theme...
    this.lockedtheme = true;
  }
  else
  {
    // Environment doesn't specify a known theme name.
    // So, the user is allowed to choose their theme.

    // Set the theme based on saved preference, system setting, or environment...
    const isDark = storedTheme ? storedTheme === 'dark' : prefersDark;
    this.setTheme(isDark);

    // Unlock the theme...
    this.lockedtheme = false;
  }

  console.log("ColorThemingService-" + this.instanceId + ":_loadTheme - ended.");
}

//#endregion



//#region Public Methods

// Public method to initialize color theme service.
// This method created to preload the service and baseline color theme.
Init()
{
```

```
  console.log("ColorThemingService-" + this.instanceId + ":Init - triggered.");

  // We don't need to actually call the _loadTheme() method, as the constructor already did.
}

// Public method to toggle the current theme light->dark or dark->light.
toggleTheme()
{
  console.log("ColorThemingService-" + this.instanceId + ":toggleTheme - triggered.");

  // See if the theme is locked...
  if(this.lockedtheme === true)
  {
    // Theme is locked by environment.
    // Don't allow user to change it.

    console.log("ColorThemingService-" + this.instanceId + ":toggleTheme - theme locked. Cannot change.");

    return;
  }

  this.setTheme(!this.isDarkThemeSubject.value);
}

// Public method call to set the desired theme.
setTheme(isDark: boolean)
{
  console.log("ColorThemingService-" + this.instanceId + ":setTheme - triggered.");

  // See if the theme is locked...
  if(this.lockedtheme === true)
    {
      // Theme is locked by environment.
      // Don't allow user to change it.

      console.log("ColorThemingService-" + this.instanceId + ":setTheme - theme locked. Cannot change.");

      return;
    }
```

```
    this.isDarkThemeSubject.next(isDark);
    localStorage.setItem('theme', isDark ? 'dark' : 'light');

    if (isDark)
    {
      document.body.classList.add('dark-theme');

      console.log("ColorThemingService-" + this.instanceId + ":setTheme - theme changed to dark.");
    }
    else
    {
      document.body.classList.remove('dark-theme');

      console.log("ColorThemingService-" + this.instanceId + ":setTheme - theme changed to light.");
    }
  }

  //#endregion
}
```

The above service is a production-ready implementation that provides public methods to set the theme or toggle it.
It also includes an Init() method that needs to be called on app startup.

See the next section for how to do this.

App.Module.ts Changes

Here's what app.module.ts will look like with the necessary pieces in it:

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

import { APP_INITIALIZER } from '@angular/core';

// Import app-level libraries...
import { LibOgaWebuiSharedkernelModule, Runtime_Sleep } from 'lib-oga-webui-sharedkernel';

import { AppComponent } from './app.component';

import { ColorThemingService } from './color-theming/services/color-theming.service';
```

```
// Import environment variable elements...
import { ENVIRONMENT_INJTOKEN, IEnvironment } from 'lib-oga-webui-sharedkernel';
import { environment } from '../environments/environment';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';



@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    AppRoutingModule,
    LibOgaWebuiSharedkernelModule,
  ],
  providers:
  [
    // Here, we register a provider that makes our app-level environment config available for consumption.
    // See this:
https://oga.atlassian.net/wiki/spaces/~311198967/pages/131760134/Angular+Expose+Environment+Config+to
+DI
    // Register our injection token for the environment class data of our app, so libraries can use it through DI...
    { provide: ENVIRONMENT_INJTOKEN, useValue: environment },
    {
      provide: APP_INITIALIZER,
      useFactory: initializeApp,
      multi: true,
      deps: [ColorThemingService, ENVIRONMENT_INJTOKEN]
    },
    provideAnimationsAsync(),
  ],
  bootstrap: [AppComponent]
})
export class AppModule {

  //#region Private Fields

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

  //#endregion
```

```
  //#region ctor / dtor

  constructor()
  {
    AppModule.lastusedInstanceId++;
    this.instanceId = AppModule.lastusedInstanceId;
    console.log("AppModule-" + this.instanceId + ":constructor - triggered.");
  }

  //#endregion
}



// This method is called when the app first starts up.
// It will load any initial config, and setup anything required.
export function initializeApp(_ctsvc:ColorThemingService,
                              env: IEnvironment): () => Promise<boolean>
{
  console.log("AppModule-" + "_" + ":initializeApp - triggered.");

  // To ensure that dependencies are stood up and ready, we must wrap our logic in a lambda, instead of naked execution...
  let lambda = async (): Promise<boolean> => {

    console.log("AppModule-" + "_" + ":initializeApp - inside startup promise...");

    // Pause a bit...
    {
      console.log("AppModule-" + "_" + ":initializeApp - Sleeping...");

      await Runtime_Sleep(1000);

      console.log("AppModule-" + "_" + ":initializeApp - Sleep done.");
    }

    // Preload the color theme service...
    {
      console.log("AppModule-" + "_" + ":initializeApp - Standup the color theme service...");
```

```
      // Verify the color theme service exists...
      if(_ctsvc == null)
      {
        console.error("AppModule-" + "_" + ":initializeApp - color theme service instance is null.");

        // Cannot load initial color theme.
        return false;
      }

      let val2 = await _ctsvc.Init();

      console.log("AppModule-" + "_" + ":initializeApp - Color Theme service preloaded.");
    }

    // Register any singletons...



    // Stand up any local config...



    // Report success...
    console.log("AppModule-" + "_" + ":initializeApp - returned.");
    return true;
  };

  return lambda;
}
```

The initializeApp() method is what initializes up the color theme service.

This method is called by an APP_INITIALIZER provider.
You can follow the above provider declaration for how to include it.

User Theme Changes

To allow a user to change light/dark mode, requires the following.

1. Import the color theme service to the component.
2. Add the color theme service to the component's constructor.
3. Call the toggleTheme() method of the service.

Here's what a minimal component would look like:

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

import { ColorThemingService } from '../../color-theming/services/color-theming.service';

@Component({
  selector: 'app-layout',
  templateUrl: './app-layout.component.html',
  styleUrl: './app-layout.component.scss'
})
export class AppLayoutComponent {

  constructor(private ctsvc: ColorThemingService)
  {
  }

  toggleTheme()
  {
    this.ctsvc.toggleTheme();
  }
}
```

The above component includes a toggle method that a button can call.
When pressed, it will tell the color theme service to swap modes.

Revision #1
Created 4 March 2025 23:19:21 by glwhite
Updated 5 March 2025 00:50:38 by glwhite