# Theming

# Agenda

Styling Angular Components

Comparing Angular Material & Bootstrap

Theming Overview

Building a Reusable Theme

Define Alternative Themes

Use Fore- and Back-Color

Use Theme Mixins

Theming Custom Components

# Styling Angular Components

# Styles in Angular

In addition to classic CSS Options there are several choices available:

- [ngStyle]
  - Allows evaluation of Expressions & methods
- [className]
  - Allows dynamic binding of classes, just like ngStyle
- Property Binding of Styles
  - `[style.color]="getColor()“`
  - `[style.background-color]="status=='error' ? 'red': 'blue'">`

# Global Styles / Component Styles

## Global Styles

- Registered in angular.json styles section - Used for own and vendor styles
- stylePreprocessorOptions allows a reference from Components without specifying the whole path
- style.css is your default global style sheet

## Component Styles

- Inline style &  External Style
- Available for a specific Component only
- Defined in the Component css

```
"styles": [
  "src/styles.scss",
  "node_modules/prismjs/themes/prism-okaidia.css"
],
"stylePreprocessorOptions": {
  "includePaths": [
    "src/theme"
  ]
},
```

# Angular Pseudo Styles

- :host
  - Style the component custom HTML element itself, and not something inside its template
- :host-context
  - Use this if you want to have a component apply a style to some element outside of it

- Just in case you discover ::ng-deep – Do not use it! Deprecated in ng10

# View Encapsulation

View Encapsulation in Angular defines how the styles defined in the template affects the other parts of the application

Angular uses Shadow DOM Concept for its Components

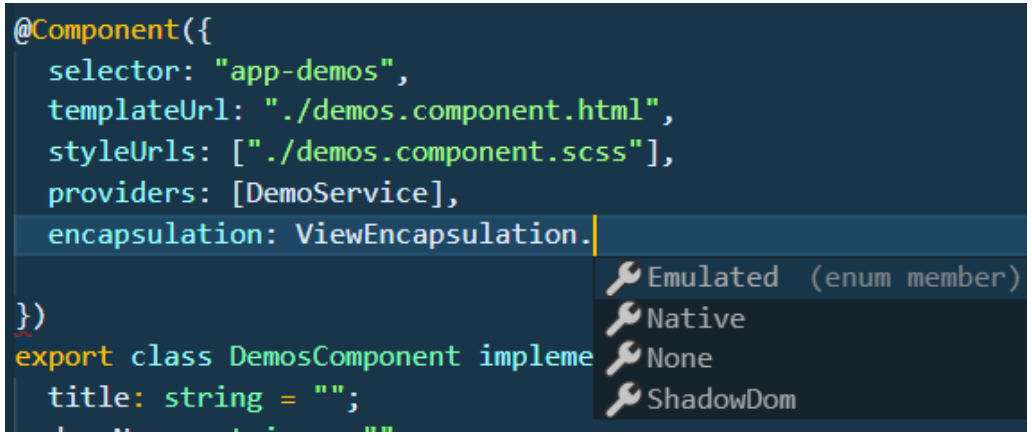
Default Behavior in Angular for Components

- Each Component is its own Shadow DOM Root
- Enables: Global Styles vs Component Styles

Native support by

- Chrome 53+
- Safari 10+

Emulation is default



# Comparing Angular Material & Bootstrap

# What is Angular Material

UI Framework that implements Googles Material Design Specification

Build by Angular Team

- Updates Released at same time as Angular

Consists of:

- Material Desing Components
- Component Dev Kit

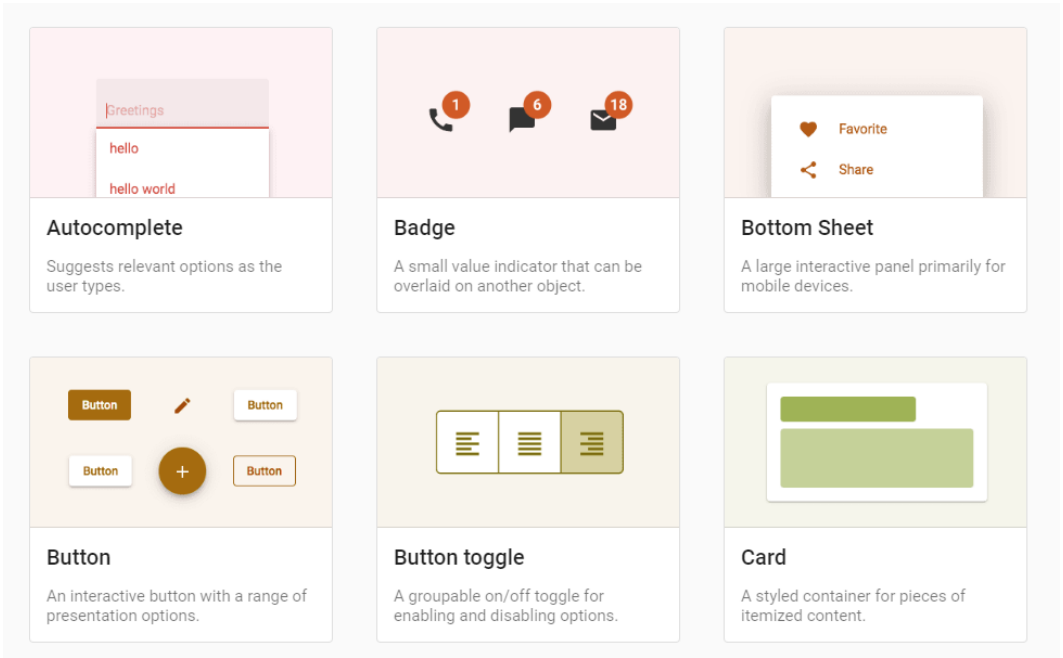
Provides Theming Support

Documentation @ https://material.angular.io/

# Material Desing Components

Includes:

- Form Controls inkluding Datepicker
- Navigation: Menu, Sidebar, Toolbar, ...
- Buttons & Indicators
- Dialog, Tooltip & Snackbar
- Data Table, Tree
- Expansion Panel, Stepper, Tabs
- ...



# Component Dev Kit (CDK)

The CDK is a set of behavior primitives for building UI components.

| | | |
|---|---|---|
| **Accessibility**<br>Utilities for screen readers, focus and more. | **Bidirectionality**<br>Utilities to respond to changes in LTR/RTL layout direction. | **Clipboard**<br>Helpers for working with the system clipboard. |
| **Drag and Drop**<br>Directives enabling drag-and-drop interactions | **Layout**<br>Utilities to respond to changes in viewport size. | **Observers**<br>Utilities to respond to changes to element properties. |
| **Overlay**<br>Utilities for dynamically displaying floating content. | **Platform**<br>Provides information about the user's platform. | **Portal**<br>Utilities for dynamically displaying content into a target. |
| **Scrolling**<br>Directives for managing scroll events. | **Stepper**<br>Presents content as steps through which to progress. | **Table**<br>A configurable component for displaying tabular data. |
| **Component Harnesses**<br>Foundation for component test harnesses. | **Text field**<br>Utilities for working with text input fields. | **Tree**<br>Presents hierarchical content as an expandable tree. |

# Themes

Angular Material comes prepackaged with several pre-built theme css files

- deeppurple-amber.css
- indigo-pink.css
- pink-bluegrey.css
- purple-green.css

Themes can use custom Colors

```
@import '~@angular/material/theming';

@include mat-core();

$app-primary: mat-palette($mat-brown, 400);
$app-accent: mat-palette($mat-grey, 400);
$app-warn: mat-palette($mat-red);
$app-fg: mat-palette($mat-grey, 200);
$app-bg: mat-palette($mat-brown, 100);

$app-theme: mat-light-theme($app-primary, $app-accent, $app-warn);

@include angular-material-theme($app-theme);
```

# Bootstrap

Bootstrap is a popular CSS Framework for developing responsive and mobile-first websites.

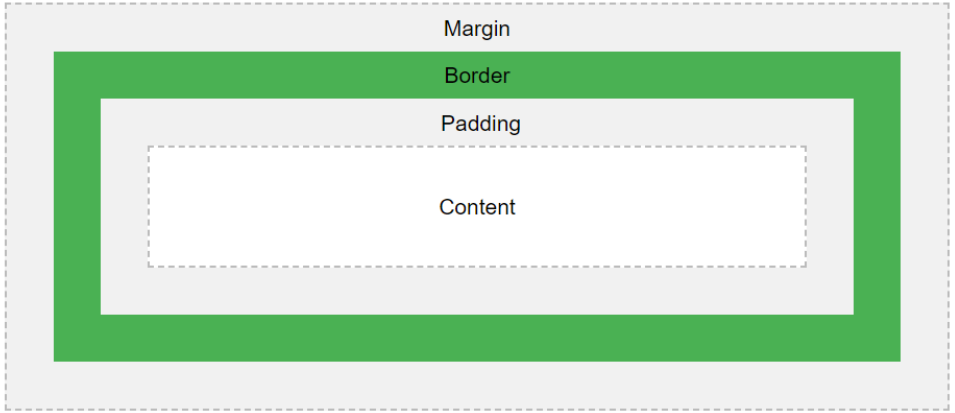
Different Parts of Bootstrap are:

- Reboot  (CSS Reset)
  - Removes Browser inconsistencies
- Typography
- Layout, Grid
- Components
- Utilities

# Reboot - CSS Reset

Reboot, a collection of element-specific CSS changes in a single file, kickstart Bootstrap to provide an elegant, consistent, and simple baseline to build upon

Resets browser specific margings, paddings, …



# Bootstrap Grid – a Flexbox Grid System

- Containers provide a means to center and horizontally pad your site's contents. Use .container for a responsive pixel width or .container-fluid for width: 100% across all viewport and device sizes.
- Rows are wrappers for columns. Each column has horizontal padding (called a gutter) for controlling the space between them.
- Column classes indicate the number of columns you'd like to use out of the possible 12 per row

| | **Extra small** <576px | **Small** ≥576px | **Medium** ≥768px | **Large** ≥992px | **Extra large** ≥1200px |
|---|---|---|---|---|---|
| **Max container width** | None (auto) | 540px | 720px | 960px | 1140px |
| **Class prefix** | `.col-` | `.col-sm-` | `.col-md-` | `.col-lg-` | `.col-xl-` |

# Bootstrap Utilities

Bootstrap provides Utilities for

- Layout (display, flexbox, margin & paddings, visibility toggles)
- Borders
- Spacings
- Images
- Position

- `t` - for classes that set `margin-top` or `padding-top`
- `b` - for classes that set `margin-bottom` or `padding-bottom`
- `l` - for classes that set `margin-left` or `padding-left`
- `r` - for classes that set `margin-right` or `padding-right`
- `x` - for classes that set both `*-left` and `*-right`
- `y` - for classes that set both `*-top` and `*-bottom`
- blank - for classes that set a `margin` or `padding` on all 4 sides of the element

# Material vs Bootstrap

- × CSS Reset
- × CSS Layout
- × CSS Utils
- ✓ Components Rich API

- ✓ CSS Reset
- ✓ CSS Layout
- ✓ CSS Utils
- × Components Rich API

# Integrating Material and Bootstrap

Install Material using ng add @angular/material

Add the following Elements from Bootstrap to style.scss

```
@import '~bootstrap/scss/functions’;
@import '~bootstrap/scss/variables’;
@import '~bootstrap/scss/mixins’;

@import '~bootstrap/scss/reboot’;
@import '~bootstrap/scss/grid’;
@import '~bootstrap/scss/type’;
@import '~bootstrap/scss/utilities’;
```

Add the Material Theming

```
@import '~@angular/material/theming';
```

# Material Theming Overview

# What is Theming

Material Theming refers to the customization of your Material Design app to better reflect your product's brand

Elements of Theming:

- Theme Colors
- Mixins
- Alternate Theme (Light / Dark)
- Standard Controls Theming
- Custom Controls Theming
- Theme Detection

# Theme Colors

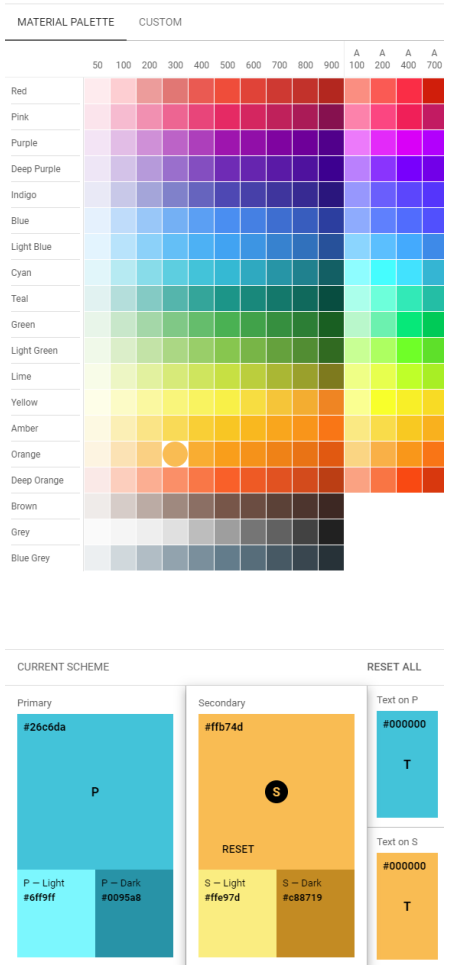
Theme Colors can be composed using Theme Tool

Angular basically knows 5 Theme Colors

- Primary
- Accent
- Warn (often used for other purposes)
- Foreground & Background

Colors ca be picked using Material Color Tool

- https://material.io/resources/color/



# Light / Dark Themes

Influence the following Elements

- Background
- Font Color

Can be detected by some OS (Mac / iOS / Windows 10) using:

- @media (prefers-color-scheme: dark)

# Color Helpers

Material contains some Color Helpers

- mat-palette
  - Provides variations of theme colors using a number
- mat-get
  - Allows access to the theme colors
- mat-color:
  - Use to set the main colors of the theme

```scss
$app-primary: mat-palette($mat-brown, 400);
$app-accent: mat-palette($mat-grey, 400);
```

```scss
@mixin ux-split-theme($theme) {
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);

  .split-title {
    mat-toolbar {
      background-color: mat-color($primary, 200);
    }
  }

  .split-main,
  .split-sidebar {
    background-color: mat-color($accent);
  }
}
```

# Sass Recap: Mixin

Mixins allow you to define styles that can be re-used throughout your stylesheet

Mixins are defined using the @mixin at-rule, which is written

- @mixin <name> { ... } or
- @mixin name(<arguments...>) { ... }

Mixins are included into the current context using the @include at-rule

```
@mixin classic-theming-theme($theme) {
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);
```

# Implementing Theming

# Elements of Theming

Theme Import

- @import "~@angular/material/theming";

Mixin for common styles

- @include mat-core();

Colors

Theme Building (light / dark)

- $app-theme: mat-light-theme($app-primary, $app-accent, $app-warn);
- $app-theme: mat-dark-theme($app-primary, $app-accent, $app-warn);

# Theme Definition

```scss
@import "~@angular/material/theming";

@include mat-core();

$app-primary: mat-palette($mat-brown, 400);
$app-accent: mat-palette($mat-grey, 400);
$app-warn: mat-palette($mat-red);
$app-fg: mat-palette($mat-grey, 200);
$app-bg: mat-palette($mat-brown, 100);

$app-theme: mat-light-theme($app-primary, $app-accent, $app-warn);

@include angular-material-theme($app-theme);
```

Import Theming support

Include common styles

Define Colors

Create the Theme

Add the theme

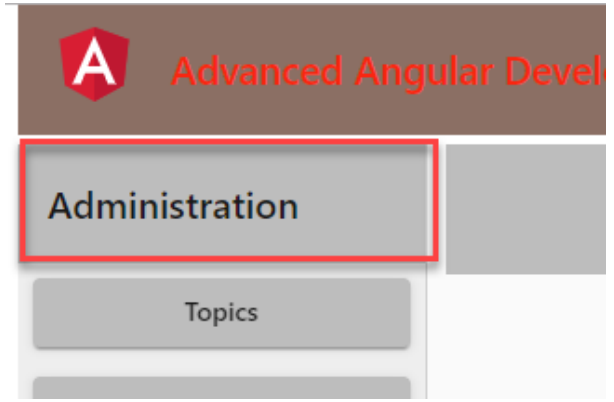
# Using Colors

By default all Components appear in neutral Color

Colors can be set

- Directly on the Component
- As a Theme Default using Tag Overrides
- Using Mixins

```
<mat-toolbar color="accent">
  <mat-toolbar-row>
    Administration
  </mat-toolbar-row>
</mat-toolbar>
```



# Theme Default using Tag Overrides

Overrides can be applied in Global Styles

Can use Theme Colors / Other Style Vars

Can override Tag / Stlye Class

- Easy to understand / implement
- Sometimes not a choice ☺

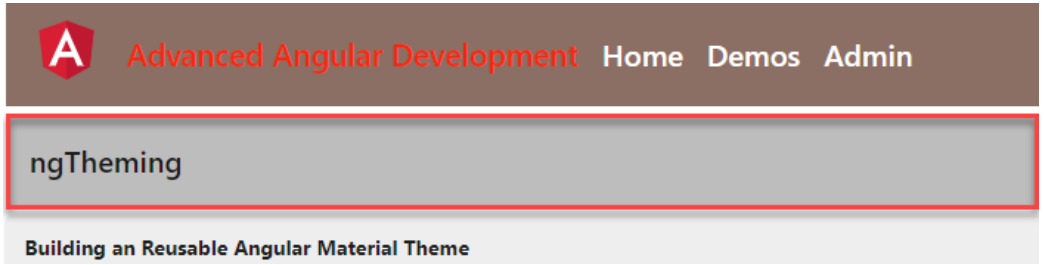
```scss
mat-card {
  background-color: mat-color($app-fg);
  margin-bottom: $big-gap;
  padding: 0 !important;
}

.mat-card-header {
  background-color: $medium-col;
  min-height: $medium-toolbar;
  padding-top: $small-gap;
}
```

# @Mixin based Overrides

Alternative approach to Tag / Style based Overrides

Good practice to have in a seperate file

Needs Theme imports




```
@mixin toolbar-theme($theme) {
  $accent: map-get($theme, accent);

  .mat-toolbar {
    background-color: mat-color($accent, 400);
  }
}
```

# Switching Theme Colors

Two Approaches:

- Implement multiple themes in one file
  - Theme is switched using class on div
  - Issues when using theme cols directly in component using mat-color
- Implement each theme in seperate files
  - Theme file as such is replaced by other file

```
$app-primary: mat-palette($mat-brown, 400);
$app-accent: mat-palette($mat-grey, 400);
$app-warn: mat-palette($mat-red);
$app-fg: mat-palette($mat-grey, 200);
$app-bg: mat-palette($mat-brown, 100);

$app-theme: mat-light-theme($app-primary, $app-accent, $app-warn);

@include angular-material-theme($app-theme);

.dark {
  $dark-primary: mat-palette($mat-teal, 800);
  $dark-accent: mat-palette($mat-lime, 900);
  $dark-warn: mat-palette($mat-red);
  $dark-fg: mat-palette($mat-grey, 200);
  $dark-bg: mat-palette($mat-lime, 800);

  $dark-theme: mat-dark-theme($dark-primary, $dark-accent, $dark-warn);
  @include angular-material-theme($dark-theme);
}
```

# Theming Components

# Re-Usable Controls

Allow encapsulation of one or more other Controls / Tags

- -> Hide complexity
- -> Allow Reusability
- -> Enable fast change

Examples:

- ux-button
  - A mat-raised-button implementation with text and icon that hides the text on smaller screen
- ux-split
  - A split with Main Area and a Sidebar that can be use in views and popups

# Theming Components

1) Add a custom class to a container surrounding  your component

2) Add base styles to the *.scss of the component EXCLUDING colors

3) Add a *-theme mixin with the $theme as param that sets the colors

4) Include the mixin and pass your current theme as param

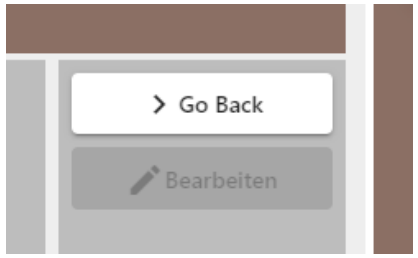
```scss
@mixin classic-theming-theme($theme) {
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);

  .classic-theming {
    background-color: mat-color($accent, 50);
    color: mat-color($accent, 900);
    border-color: mat-color($primary, 100);
  }
}

@include classic-theming-theme($app-theme);
```

# ux-button

Just a simple component

fxHide.lt-lg hides span with thext

@Inputs / @Outputs allow binding & events




```
<ux-button
  [label]="'Bearbeiten'"
  [icon]="'edit'"
  [disabled]="isDisabled"
></ux-button>
```

ux-button.component.html

src > app > ux > ux-button > ux-button.component.html > ...

```
<button mat-raised-button (click)="buttonClicked()" [disabled]="disabled">
  <mat-icon>{{ icon }}</mat-icon>
  <span fxHide.lt-lg>{{ label }}</span>
</button>
```

ux-button.component.ts

src > app > ux > ux-button > ux-button.component.ts > ...

```
import { Component, OnInit, Input, Output, EventEmitter } from "@angular/core";

@Component({
  selector: "ux-button",
  templateUrl: "./ux-button.component.html",
  styleUrls: ["./ux-button.component.scss"]
})
export class uxButtonComponent implements OnInit {
  @Input() disabled: boolean = false;
  @Input() label: string = "";
  @Input() icon: string;
  @Output() click: EventEmitter<void> = new EventEmitter();

  constructor() {}

  ngOnInit() {}

  buttonClicked() {
    this.click.emit(null);
  }
}
```

# ux-split

A Component using Content Projection

Very handy if you need a Split your App

- Change once if needed
- Uses CSS Grid (Flex Layout Implementation)


ux-split.component.html

src > app > ux > ux-split > ux-split.component.html > div.container

```
<div
  gdGap="0.5rem" gdAreas="title title | main toolbar" gdColumns="800px auto"
  gdRows="60px auto" class="container"
>
  <div gdArea="title" class="split-title">
    <mat-toolbar mat-dialog-title>
      <mat-toolbar-row>
        <ng-content select=".title"></ng-content>
      </mat-toolbar-row>
    </mat-toolbar>
  </div>
  <div gdArea="main" class="split-main">
    <ng-content select=".main"></ng-content>
  </div>
  <div gdArea="toolbar" class="split-sidebar">
    <ng-content select=".toolbar"></ng-content>
  </div>
</div>
```

ux-split.component.ts

src > app > ux > ux-split > ux-split.component.ts > ...

```
import { Component, OnInit } from "@angular/core";

@Component({
  selector: "ux-split",
  templateUrl: "./ux-split.component.html",
  styleUrls: ["./ux-split.component.scss"]
})
export class uxSplitComponent implements OnInit {
  constructor() {}

  toolbar = "100px";

  ngOnInit() {}
}
```

# Theming Re-Usable Controls

Custom Control Theming is done using Mixins

Use mat-get() to get a reference to the theme colors


```scss
@mixin ux-spilt-theme($theme) {
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);

  .split-title {
    mat-toolbar {
      background-color: mat-color($primary);
    }
  }

  .split-main,
  .split-sidebar {
    background-color: mat-color($accent);
  }
}
```

# Simple Approach using Bootstrap classes

Just include and use Bootstrape Utility classes

Add them to containers (divs, ng-container)

Implement a Mixin to use theme colours

```
<div class="border m-5 p-5 font-weight-bold my-text my-background">
  <h3>Alternative Theming</h3>
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Sunt totam, fugiat
  voluptate ratione recusandae molestias fugit est temporibus tempore rem ex
  doloremque adipisci. Assumenda nihil dicta possimus nulla alias repudiandae?
  Quo, eos labore fugiat nemo molestias, iure placeat minus, quae inventore est
```

```
@mixin set-colors($theme) {
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);

  .my-text {
    color: mat-color($primary);
  }

  .my-background {
    background-color: mat-color($accent);
  }
}

@include set-colors($app-theme);
```

# Supporting Multible Themes

Several approches for supporting multible Themes:

- Div Container with [ngClass]
  - Works fine as long as you use color Directive. ie: color="accent“
  - Does not work well with mixins that reference colors
- Loading and attaching theme using Code
  - Load theme using a custom “StyleService”
  - Sample see ngThemeSwitcher

```
export class StyleManagerService {

  constructor() { }

  setStyle(key: string, href: string) {
    getLinkElementForKey(key).setAttribute('href', href);
  }

  removeStyle(key: string) {
    const existingLinkElement = getExistingLinkElementByKey(key);
    if (existingLinkElement) {
      document.head.removeChild(existingLinkElement);
    }
  }
}
```