Enhancing the E-commerce Application with Component Styling

Introduction

In this example, we'll enhance the previously created e-commerce application by incorporating the concepts from the https://angular.dev/guide/components/styling. We'll focus on: - Component-specific styles - View encapsulation modes - Host selectors - Style binding - Conditional styling - Shadow DOM usage

This will provide a realistic and complex enough application to see these concepts in action.

Application Overview

Our enhanced e-commerce application will include the following components: 1) `AppComponent`: The root component. 2) `ProductListComponent`: Displays a list of products. 3) `ProductItemComponent`: Displays an individual product. 4) `CartComponent`: Displays the shopping cart. 5) `ProductService`: Handles fetching and managing product data.

We'll focus on styling these components to demonstrate various Angular styling techniques.

Code

app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <header class="app-header">
      <h1>E-commerce Store</h1>
      <app-cart [cartItems]="cartItems"></app-cart>
    </header>
    <main>
      <app-product-list (addToCart)="handleAddToCart($event)"></app-product-list>
    </main>
  `,
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.None, // Using None for global styles
})
export class AppComponent {
  cartItems: any[] = [];

  handleAddToCart(product: any): void {
    this.cartItems.push(product);
  }
}
```

app.component.css

```
.app-header {
  background-color: #1976d2;
  color: white;
  padding: 1rem;
}

main {
  padding: 1rem;
  background-color: #f5f5f5;
}
```

product-list.component.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
import { ProductService } from '../product.service';

@Component({
  selector: 'app-product-list',
  template: `
    <h2>Products</h2>
    <div class="product-list">
      <app-product-item
        *ngFor="let product of products"
        [product]="product"
        (add)="onAddToCart($event)"
        [class.featured]="product.featured"
      ></app-product-item>
    </div>
  `,
  styleUrls: ['./product-list.component.scss'], // Using SCSS for nested styles
})
export class ProductListComponent implements OnInit {
  products: any[] = [];
  @Output() addToCart = new EventEmitter<any>();

  constructor(private productService: ProductService) {}

  ngOnInit(): void {
    this.products = this.productService.getProducts();
  }

  onAddToCart(product: any): void {
    this.addToCart.emit(product);
  }
}
```

product-list.component.scss

```
.product-list {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;

  ::ng-deep app-product-item {
    flex: 1 1 calc(33% - 1rem);
    box-sizing: border-box;
  }
}

h2 {
  color: #424242;
}
```

product-item.component.ts

```
import {
  Component,
  Input,
  Output,
  EventEmitter,
  OnChanges,
  SimpleChanges,
  ChangeDetectionStrategy,
  ViewEncapsulation,
  HostBinding,
} from '@angular/core';

@Component({
  selector: 'app-product-item',
  template: `
    <div class="product-image" [ngStyle]="{ 'background-image': 'url(' + product.image
        + ')' }"></div>
    <h3>{{ product.name }}</h3>
    <p>{{ product.description }}</p>
    <p class="price">{{ product.price | currency }}</p>
    <button (click)="addToCart()" [disabled]="productOutOfStock">Add to Cart</button>
    <ng-content></ng-content>
  `,
  styleUrls: ['./product-item.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
  encapsulation: ViewEncapsulation.Emulated, // Default encapsulation
})
export class ProductItemComponent implements OnChanges {
  @Input() product: any;
  @Output() add = new EventEmitter<any>();

  @HostBinding('class.out-of-stock') productOutOfStock = false;

  ngOnChanges(changes: SimpleChanges): void {
    if (changes['product']) {
      this.productOutOfStock = this.product.stock === 0;
    }
  }

  addToCart(): void {
    this.add.emit(this.product);
  }
}
```

product-item.component.css

```
:host {
  display: block;
  border: 1px solid #e0e0e0;
  padding: 1rem;
  background-color: white;
  position: relative;
}

.product-image {
  width: 100%;
  height: 200px;
  background-size: cover;
  background-position: center;
}

.price {
  font-weight: bold;
  color: #388e3c;
}

button {
  background-color: #1976d2;
  color: white;
  border: none;
  padding: 0.5rem 1rem;
  cursor: pointer;
}

button[disabled] {
  background-color: #9e9e9e;
  cursor: not-allowed;
}

:host(.out-of-stock) {
  opacity: 0.6;
}

:host-context(.featured) {
  border-color: #ffd54f;
}
```

cart.component.ts

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

@Component({
  selector: 'app-cart',
  template: `
    <h2>Shopping Cart</h2>
    <div *ngIf="cartItems.length > 0; else emptyCart">
      <div *ngFor="let item of cartItems">
        {{ item.name }} - {{ item.price | currency }}
      </div>
      <p class="total">Total: {{ getTotal() | currency }}</p>
    </div>
    <ng-template #emptyCart>
      <p>Your cart is empty.</p>
    </ng-template>
  `,
  styles: [
    `
      :host {
        display: block;
        padding: 1rem;
        background-color: #fffde7;
      }
      .total {
        font-weight: bold;
        margin-top: 1rem;
      }
    `,
  ],
})
export class CartComponent {
  @Input() cartItems: any[] = [];

  getTotal(): number {
    return this.cartItems.reduce((total, item) => total + item.price, 0);
  }
}
```

product.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class ProductService {
  private products = [
    {
      id: 1,
      name: 'Laptop',
      description: 'A high-performance laptop',
      price: 1299.99,
      image: 'assets/images/laptop.jpg',
      stock: 5,
      featured: true,
    },
    {
      id: 2,
      name: 'Smartphone',
      description: 'A powerful smartphone',
      price: 799.99,
      image: 'assets/images/smartphone.jpg',
      stock: 0,
      featured: false,
    },
    {
      id: 3,
      name: 'Headphones',
      description: 'Noise-cancelling headphones',
      price: 199.99,
      image: 'assets/images/headphones.jpg',
      stock: 12,
      featured: true,
    },
  ];

  getProducts(): any[] {
    return this.products;
  }
}
```

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { ProductListComponent } from './product-list/product-list.component';
import { ProductItemComponent } from './product-item/product-item.component';
import { CartComponent } from './cart/cart.component';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductItemComponent,
    CartComponent,
  ],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Explanation of the Code

AppComponent

- **Purpose**: Acts as the root component of the application.
- **Template**:
  - Displays the application header with the title.
  - Includes the `<app-cart>` component and passes the `cartItems` array via property binding.
  - Includes the `<app-product-list>` component and binds to its `addToCart` event.
- **Styles** (`app.component.css`):
  - Styles the header with a blue background and white text.
  - Styles the main content area with padding and a light gray background.
- **View Encapsulation**:
  - Uses `ViewEncapsulation.None` to apply styles globally, affecting child components if selectors match.

ProductListComponent

- **Purpose**: Displays a list of products fetched from the `ProductService`.
- **Template**:
  - Uses a `<div>` with the class `product-list` to wrap the products.
  - Iterates over the `products` array using `*ngFor` and renders a `<app-product-item>` for each product.
  - Binds each product to the `[product]` input property of `ProductItemComponent`.
  - Captures the add event from `ProductItemComponent` and calls `onAddToCart()`.
  - Conditionally adds the `featured` class to `app-product-item` if the product is featured.
- **Styles** (`product-list.component.scss`):
  - Uses SCSS for nested styles.
  - Styles the `.product-list` to display products in a flexible grid layout.
  - Uses `::ng-deep` to apply styles to child components deeply, affecting their internal elements.
- **Logic**:
  - Fetches products from `ProductService` in `ngOnInit()`.
  - Emits an `addToCart` event when a product is added.
- **Annotations**:
  - `@Output() addToCart`: Event emitter to notify the parent component when a product is added.
  - `styleUrls`: Uses an SCSS file for styles.

ProductItemComponent

- **Purpose**: Displays individual product details and allows adding the product to

the cart.
- **Template**:
  - Displays the product image using `[ngStyle]` for inline styling.
  - Displays the product name, description, and price.
  - The “Add to Cart” button is disabled if the product is out of stock.
  - Uses `<ng-content>` to project additional content if needed.
- **Styles** (`product-item.component.css`):
  - Uses the `:host` selector to style the component’s root element.
  - Styles the product container, image, price, and button.
  - Uses `[disabled]` selector to style the disabled state of the button.
  - Applies the `.out-of-stock` class to the host element based on the product’s stock status.
  - Uses `:host-context(.featured)` to style the component differently if it’s a featured product.
- **Logic**:
  - Uses `ngOnChanges()` to update the `productOutOfStock` flag when the `product` input changes.
  - Emits an `add` event when the “Add to Cart” button is clicked, unless the product is out of stock.
- **Annotations**:
  - `@Input() product`: Receives product data.
  - `@Output() add`: Event emitter to notify when a product is added.
  - `@HostBinding('class.out-of-stock')`: Binds the `out-of-stock` class to the host element based on the product’s stock.
  - `encapsulation: ViewEncapsulation.Emulated`: Uses default encapsulation to scope styles to the component.

CartComponent

- **Purpose**: Displays the items added to the shopping cart.
- **Template**:
  - Displays a message if the cart is empty using `<ng-template>`.
  - Iterates over the `cartItems` array using `*ngFor` and displays each item’s name and price.
  - Displays the total price of the items in the cart.
- **Styles**:
  - Uses inline styles to style the component.
  - Styles the component’s host element with padding and a light yellow background.
  - Styles the total price with bold text.
- **Logic**:
  - Receives `cartItems` from the parent component via `@Input()`.
  - Calculates the total price using the `getTotal()` method.

ProductService

- **Purpose**: Provides product data to components.
- **Logic**:
  - Defines a `products` array with additional properties like `image`, `stock`, and `featured`.
  - Implements `getProducts()` method to return the list of products.
- **Annotations**:
  - `@Injectable({ providedIn: 'root' })`: Makes the service available application-wide.

AppModule

- **Purpose**: The root module that bootstraps the application.
- **Declarations**:

    - Lists all components used in the application.
- **Imports**:
    - `BrowserModule` is imported to run the app in a browser.
- **Bootstrap**:
    - Bootstraps the `AppComponent`.

Key Angular Styling Concepts Demonstrated

Component Styles

- **Component-specific styles**: Each component has its own styles defined in separate CSS or SCSS files, allowing for modular and maintainable styling.
- **Global styles**: `AppComponent` uses `ViewEncapsulation.None` to apply styles globally, affecting all components if selectors match.

View Encapsulation Modes

- **Emulated**: Used in `ProductItemComponent`, where styles are scoped to the component using Angular's default encapsulation strategy.
- **None**: Used in `AppComponent` to apply styles globally, which can be useful for shared styles across multiple components.

Host Selectors

- `:host`: Used in `ProductItemComponent` to style the component's root element, such as setting the display and border
- `:host-context()`: Used to apply styles based on a condition in the component's ancestor, such as styling featured products differently.

Style Binding

- `[ngStyle]`: Used in `ProductItemComponent` to dynamically set the background image of the product based on the product data.

Conditional Styling

- **Class binding**: Used in `ProductListComponent` to conditionally add the `featured` class to `app-product-item` elements.
- **Host binding**: Used in `ProductItemComponent` to apply the `out-of-stock` class based on the product's stock status.

Shadow DOM Usage

- **Encapsulation**: Demonstrated through the use of different encapsulation strategies (`Emulated` and `None`) to control how styles are applied and scoped within components.

SCSS for Nested Styles

- **SCSS**: Used in `ProductListComponent` to demonstrate nested styles and the use of `::ng-deep` for styling child components deeply.

Angular Pipes

- **Currency pipe**: Used in `ProductItemComponent` and `CartComponent` to format product prices, demonstrating the use of Angular pipes for data transformation.

Conclusion

This enhanced e-commerce application demonstrates various Angular styling techniques, including component-specific styles, view encapsulation, host selectors, style binding, and conditional styling. By incorporating these concepts, the application becomes more modular, maintainable, and visually appealing. The use of SCSS and Angular pipes further enhances the styling capabilities and data presentation within the application.