Enhanced E-commerce Example with Custom Projection with ng-content

Introduction

This example builds on the logic found in Anatomy of a Component and demonstrates more in-depth use of Angular Custom Projection with ng-content. This code builds on the original e-commerce application logic, keeping the same folder structure but adding new features that showcase content projection. After the code sections, you'll find a detailed explanation of everything.

Code

product-details-wrapper.component.ts

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

/**
 * ProductDetailsWrapperComponent
 * ------------------------------
 * Purpose:
 *   Demonstrates multi-slot content projection by providing
 *   distinct areas (header, body, actions) for projected content.
 *
 * Example Usage in a parent component's template:
 *   <app-product-details-wrapper>
 *     <ng-container product-header>
 *       <!-- Header content goes here -->
 *     </ng-container>
 *     <ng-container product-body>
 *       <!-- Body / main content goes here -->
 *     </ng-container>
 *     <ng-container product-actions>
 *       <!-- Footer / action content goes here -->
 *     </ng-container>
 *   </app-product-details-wrapper>
 */
@Component({
  selector: 'app-product-details-wrapper',
  template: `
    <div class="details-wrapper">
      <header>
        <ng-content select="[product-header]"></ng-content>
      </header>
      <main>
        <ng-content select="[product-body]"></ng-content>
      </main>
      <footer>
        <ng-content select="[product-actions]"></ng-content>
      </footer>
    </div>
  `,
  styleUrls: ['./product-details-wrapper.component.css']
})
export class ProductDetailsWrapperComponent {}
```

product-item.component.ts

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

@Component({
  selector: 'app-product-item',
  template: `
    <!--
      Instead of displaying product details directly, we use our
      multi-slot content projection layout to organize the content
      based on designated slots.
    -->
    <app-product-details-wrapper>
      <!-- Project product title into the header slot -->
      <ng-container product-header>
        <h3>{{ product.name }}</h3>
      </ng-container>

      <!-- Project product description into the body slot -->
      <ng-container product-body>
        <p>{{ product.description }}</p>
        <p>Price: {{ product.price | currency }}</p>
      </ng-container>

      <!-- Project an "Add to Cart" button into the actions slot -->
      <ng-container product-actions>
        <button (click)="addToCart()">Add to Cart</button>
      </ng-container>
    </app-product-details-wrapper>
  `,
  styleUrls: ['./product-item.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProductItemComponent implements OnChanges {
  @Input() product: any;
  @Output() add = new EventEmitter<any>();

  /**
   * Detects changes made to the product input property.
   * If the `product` input changes, it logs the new value.
   */
  ngOnChanges(changes: SimpleChanges): void {
    if (changes['product']) {
      console.log('Product changed:', changes['product'].currentValue);
    }
  }

  /**
   * Emits the current product to be added to the cart.
   */
  addToCart(): void {
    this.add.emit(this.product);
  }
}
```

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';
import { ProductService } from './product.service';
import { ProductDetailsWrapperComponent } from './product-details-wrapper.component';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductItemComponent,
    CartComponent,
    ProductDetailsWrapperComponent  // <-- Added here
  ],
  imports: [
    BrowserModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

product-details-wrapper.component.css

```
.details-wrapper {
  border: 1px solid #ccc;
  padding: 1rem;
  margin: 0.5rem 0;
  background-color: #fafafa;
}

.details-wrapper header {
  background-color: #eee;
  padding: 0.5rem;
}

.details-wrapper main {
  margin: 0.5rem 0;
}

.details-wrapper footer {
  border-top: 1px solid #ccc;
  padding: 0.5rem;
  text-align: right;
}
```

Explanation of the Code

ProductDetailsWrapperComponent

- **Purpose**:
  - This component demonstrates multi-slot content projection, allowing

different parts of content to be projected into specific areas (header, body, actions) of the component template.

- **Template**:
  - Uses `<ng-content>` with `select` attributes to define slots for content projection:
    - `<ng-content select="[product-header]"></ng-content>`: Projects content marked with the `product-header` attribute.
    - `<ng-content select="[product-body]"></ng-content>`: Projects content marked with the `product-body` attribute.
    - `<ng-content select="[product-actions]"></ng-content>`: Projects content marked with the `product-actions` attribute.
- **Styling**:
  - The CSS file (`product-details-wrapper.component.css`) provides basic styling for the wrapper, header, main, and footer sections to visually separate the projected content.

ProductItemComponent

- **Purpose**:
  - Displays individual product details using the `ProductDetailsWrapperComponent` for layout, demonstrating content projection in action.
- **Template**:
  - Utilizes `<app-product-details-wrapper>` to organize content into slots:
  - `<ng-container product-header>`: Projects the product name into the header slot.
  - `<ng-container product-body>`: Projects the product description and price into the body slot.
  - `<ng-container product-actions>`: Projects the “Add to Cart” button into the actions slot.
- **Logic**:
  - Implements `ngOnChanges()` to log changes to the `product` input property.
  - Emits an `add` event when the “Add to Cart” button is clicked, passing the product data to the parent component.
- **Annotations**:
  - `@Input() product`: Receives product data from the parent component.
  - `@Output() add`: An event emitter that notifies the parent component when the product is added to the cart.
  - `changeDetection: ChangeDetectionStrategy.OnPush`: Optimizes performance by using the `OnPush` change detection strategy.

ProductItemComponent

- **Purpose**:
  - Displays individual product details and provides buttons to add the product to the cart or wishlist.
- **Template**:
  - Displays the product name and description.
  - Includes “Add to Cart” and “Add to Wishlist” buttons that trigger the respective methods.
- **Logic**:
  - Implements the `ngOnChanges()` lifecycle hook to detect changes to the `product` input.
  - Emits `add` and `wishlist` events when the respective buttons are clicked.

AppModule

- **Purpose**:
  - The root module that bootstraps the application and declares all components used in the application.
- **Declarations**:
  - Includes `ProductDetailsWrapperComponent` in the declarations array, making it

available for use in templates.
- **Imports**: `BrowserModule` is imported to run the app in a browser.
- **Providers**:
  - `ProductService` is provided at the root level for dependency injection.
- **Bootstrap**:
  - Bootstraps the `AppComponent` to launch the application.

Overall Flow

- **AppComponent**:
  - Acts as the root component, hosting `ProductListComponent` and `CartComponent`.
  - Manages the `cartItems` array and handles adding products to the cart.
- **ProductListComponent**:
  - Fetches products from `ProductService` and renders a list of `ProductItemComponent` instances.
  - Emits an `addToCart` event when a product is added, passing the product data to `AppComponent`.
- **CartComponent**:
  - Displays items added to the shopping cart, using the `currency` pipe to format prices.
- **ProductDetailsWrapperComponent**:
  - Provides a flexible layout for displaying product details, allowing different content to be projected into specific slots.

Key Angular Concepts Demonstrated

- **Content Projection**:
  - Demonstrates multi-slot content projection using `<ng-content>` with `select` attributes.
- **Component Inputs and Outputs**:
  - Uses `@Input()` to receive data and `@Output()` with `EventEmitter` to send events up the component tree.
- **Change Detection**:
  - Utilizes `ChangeDetectionStrategy.OnPush` to optimize performance by reducing unnecessary checks.
- **Modularization**:
  - Separates layout logic into `ProductDetailsWrapperComponent`, promoting reusability and separation of concerns.

Conclusion

This detailed explanation covers the purpose, template structure, logic, and key concepts of each component, providing a comprehensive understanding of how the enhanced Angular example demonstrates content projection and other Angular features.