Anatomy of a component

Introduction

Let’s create a realistic Angular application that covers the functionalities described in the https://angular.dev/guide/components. We’ll build a simple e-commerce application that includes the following features: - Component creation and nesting - Component inputs and outputs - Lifecycle hooks - Interaction between parent and child components - View encapsulation - Content projection - Change detection

Application Overview

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

Code

app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <h1>E-commerce Store</h1>
    <app-product-list (addToCart)="handleAddToCart($event)"></app-product-list>
    <app-cart [cartItems]="cartItems"></app-cart>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  cartItems: any[] = [];

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

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>
    <app-product-item
      *ngFor="let product of products"
      [product]="product"
      (add)="onAddToCart($event)"
    ></app-product-item>
  `,
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {
  products: any[] = [];
  @Output() addToCart = new EventEmitter<any>();

  constructor(private productService: ProductService) {}

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

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

product-item.component.ts

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

@Component({
  selector: 'app-product-item',
  template: `
    <div class="product">
      <h3>{{ product.name }}</h3>
      <p>{{ product.description }}</p>
      <ng-content></ng-content>
      <button (click)="addToCart()">Add to Cart</button>
    </div>
  `,
  styleUrls: ['./product-item.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProductItemComponent implements OnChanges {
  @Input() product: any;
  @Output() add = new EventEmitter<any>();

  ngOnChanges(changes: SimpleChanges) {
    if (changes['product']) {
      console.log('Product changed:', changes['product'].currentValue);
    }
  }

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

cart.component.ts

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

```
@Component({
  selector: 'app-cart',
  template: `
    <h2>Shopping Cart</h2>
    <div *ngFor="let item of cartItems">
      {{ item.name }} - {{ item.price | currency }}
    </div>
  `,
  styleUrls: ['./cart.component.css']
})
export class CartComponent {
  @Input() cartItems: any[] = [];
}
```

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
    },
    {
      id: 2,
      name: 'Smartphone',
      description: 'A powerful smartphone',
      price: 799.99
    },
    {
      id: 3,
      name: 'Headphones',
      description: 'Noise-cancelling headphones',
      price: 199.99
    }
  ];

  getProducts() {
    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 {}
```

Styles and Templates

For brevity, the styles (`.css` files) and additional templates are minimal. You can style the components as needed.

Explanation of the Code

AppComponent

- **Purpose**: Acts as the root component of the application.
- **Template**:
  - Displays the application title.
  - Includes the `<app-product-list>` component and binds to its `addToCart` event.
  - Includes the `<app-cart>` component and passes the `cartItems` array to it via property binding.
- **Logic**:
  - Maintains a `cartItems` array to keep track of products added to the cart.
  - Implements `handleAddToCart()` method to handle adding products to the cart.

ProductListComponent

- **Purpose**: Displays a list of products fetched from the `ProductService`.
- **Template**:
  - 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()`.
- **Logic**:
  - Uses `ProductService` to fetch the list of products in `ngOnInit()`.
  - Emits an `addToCart` event when a product is added, passing the product data to the parent component (`AppComponent`).
- **Annotations**:
  - `@Output() addToCart`: An event emitter that notifies the parent component when a product is added to the cart.

ProductItemComponent

- **Purpose**: Displays individual product details and allows adding the product to the cart.
- **Template**:
  - Displays the product name and description.
  - Uses `<ng-content>` for content projection (though not extensively utilized here).
  - Includes an “Add to Cart” button that triggers the `addToCart()` method.
- **Logic**:
  - Implements the `ngOnChanges()` lifecycle hook to detect changes to the `product`

    input.
  - Emits an `add` event when the "Add to Cart" button is clicked.
- **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 changing detection strategy.

CartComponent

- **Purpose**: Displays the items added to the shopping cart.
- **Template**:
  - Iterates over the `cartItems` array using `*ngFor` and displays each item's name and price.
  - Uses the Angular `currency` pipe to format the price.
- **Logic**:
  - Receives `cartItems` from the parent component via the `@Input()` property.

ProductService

- **Purpose**: Provides product data to components.
- **Logic**:
  - Defines a private `products` array containing product objects.
  - 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` to launch the application.

Key Angular Concepts Demonstrated

Components and Nesting

- Created multiple components (`AppComponent`, `ProductListComponent`, `ProductItemComponent`, `CartComponent`) and demonstrated how they nest within each other.

Component Inputs and Outputs

- Used `@Input()` to receive data (`product` in `ProductItemComponent`, `cartItems` in `CartComponent`).
- Used `@Output()` with `EventEmitter` to send events up the component tree (`addToCart` in `ProductListComponent`, `add` in `ProductItemComponent`).

Lifecycle Hooks

- Implemented `ngOnInit()` in `ProductListComponent` to fetch products when the component initializes.

- Used `ngOnChanges()` in `ProductItemComponent` to react to changes in input properties.

Interaction Between Parent and Child Components

- Parent components pass data to child components via property binding.
- Child components emit events to notify parent components of actions (like adding a product to the cart).

View Encapsulation and Change Detection

- Used `changeDetection: ChangeDetectionStrategy.OnPush` in `ProductItemComponent` to optimize performance.
- The styles are encapsulated within each component.

Content Projection

- Demonstrated `<ng-content>` in `ProductItemComponent` to potentially allow content to be projected into the component (though not fully utilized in this basic example).

Services and Dependency Injection

- Created a `ProductService` to handle data retrieval.
- Injected `ProductService` into `ProductListComponent` via the constructor.

Pipes

- Used the `currency` pipe in `CartComponent` to format product prices.

Conclusion

This example provides a basic e-commerce application demonstrating several key Angular concepts related to components. It showcases how to create reusable components, pass data between them, and manage state and events within an Angular application.