Submission: Upload your work to GitHub repo till the next practice lesson

Online Store (Continued)
Component Architecture & User Interaction

In this lab you will extend the Online Store project from Lab 4 by introducing a category-based product hierarchy, breaking the UI into well-defined Angular components, and adding interactive features such as likes and item removal. The focus is on parent–child component communication, component lifecycle, and dynamic rendering.

Learning Objectives

By the end of this lab, you will be able to:

- Organize application data into a hierarchical structure (categories → products)
- Decompose a UI into multiple Angular components with clear responsibilities
- Pass data from parent to child components using `input()` or `@Input()`
- Emit events from child to parent components using `output()` or `@Output()` and `EventEmitter`
- Dynamically render lists of components based on user selection
- Handle user interactions such as liking and deleting items
- Understand the Angular component lifecycle hooks
- Maintain clean separation of concerns across components

Prerequisites

Before starting this lab, make sure you have:

- **Completed Lab 4** — your Online Store project should be working and pushed to GitHub
- **Angular CLI** installed and the project running via `ng serve`
- A basic understanding of `input()/output()` or `@Input()/@Output()` (review the links in Useful Resources if needed)

Reminder: Angular 17+ Syntax

As introduced in Lab 4, you may use either the new or legacy syntax. The new syntax is preferred:

- **Control Flow:** `@for/@if/@else` (preferred) or `*ngFor/*ngIf`
- **Inputs:** `input()` signal function (preferred) or `@Input()` decorator
- **Outputs:** `output()` function (preferred) or `@Output()` with `EventEmitter`

Example — Input/Output comparison:

```
// Old syntax (still works)
@Input() product: Product;
```

```
@Output() delete = new EventEmitter<number>();
// New syntax (preferred)
product = input.required<Product>();
delete = output<number>();
```

Tasks (3)

1. **Define the Category → Product Hierarchy**

   Restructure your product data so that every product belongs to a category. The application should follow the hierarchy: **Category ⇒ Product Items**.

   a. Create a `Category` interface (e.g., `category.model.ts`) with at least:
      - `id: number` — unique identifier for the category
      - `name: string` — category display name (e.g., "Smartphones", "Laptops", "Headphones", "Tablets")

   b. Update your `Product` interface to include a `likes: number` field (initialized to 0) and a `categoryId: number` field

   c. Create exactly **4 categories**, each containing exactly **5 products** (20 products total)

   d. All products must still link to real items on kaspi.kz

   e. Store the data in a service (e.g., `ProductService`) or in a separate data file — do **not** hardcode arrays directly inside components

   Tip: Choose categories that make sense together, for example: Smartphones, Laptops, Headphones, and Tablets. Each should have 5 real products from kaspi.kz.

2. **Build the Component Architecture**

   Refactor your application into **3 main components** with clearly defined responsibilities:

   a. AppComponent (root component)
   - Displays the list of categories (e.g., as buttons, tabs, cards, or a sidebar navigation)
   - Tracks which category is currently selected
   - When a category is clicked, the products belonging to that category are passed to `ProductListComponent`
   - If no category is selected, display a welcome message or prompt the user to select a category

   b. ProductListComponent
   - Receives an array of products via `input()` or `@Input()`
   - Renders a `ProductItemComponent` for each product using `@for` or `*ngFor`
   - Listens for delete events from child `ProductItemComponent` and removes the product from the list
   - Displays a message (e.g., "No products in this category") if the product list is empty after deletions

   c. ProductItemComponent
   - Receives a single product object via `input()` or `@Input()`
   - Displays product details: image, name, description, price, rating, and number of likes
   - Contains a **"Like" button** — clicking it increments the likes counter for that product
   - Contains a **"Delete" button** — clicking it emits an event to the parent via `output()` or `@Output()` to remove the product
   - Retains the **"Share" button** from Lab 4 (WhatsApp / Telegram)

Note: The component hierarchy should be: AppComponent → ProductListComponent → ProductItemComponent. Data flows down via input(), and events flow up via output().

3. **Implement Interactive Features**

   a. Like Functionality

   - Each product card displays its current number of likes (e.g., "❤ 12")
   - Clicking the "Like" button increments the likes count by 1
   - The like count should update immediately in the UI
   - *Optional:* Add visual feedback (e.g., heart icon animation, color change on click)

   b. Delete Functionality

   - Each product card has a "Delete" button (e.g., a trash icon or red button)
   - Clicking "Delete" removes the product from the displayed list
   - The delete event must be emitted from `ProductItemComponent` to `ProductListComponent` using `output()` or `@Output()`
   - *Optional:* Add a confirmation dialog before deletion (e.g., "Are you sure?")

   c. Category Switching

   - Clicking a category should immediately display its products
   - Highlight or visually indicate the currently selected category
   - Use `@if` or `*ngIf` to show/hide the product list based on selection

Requirements

- **Component Structure:** Exactly 3 components — `AppComponent`, `ProductListComponent`, `ProductItemComponent`
- **Input/Output:** Data must flow from parent to child via `input()`; events (delete) must flow from child to parent via `output()`
- **Categories:** Exactly 4 categories with 5 products each (20 products total), all linked to real kaspi.kz items
- **Like Feature:** Each product must have a working like button that increments a counter displayed on the card
- **Delete Feature:** Each product must be removable; the UI must update dynamically after removal
- **TypeScript:** Use interfaces for all data models; avoid `any` type
- **CSS Styling:** Clean, scoped component styles; responsive layout using Flexbox or CSS Grid
- **Code Quality:** Meaningful names, no unused code, components should be focused and concise
- **Share Button:** Retain the WhatsApp/Telegram share functionality from Lab 4

Deliverables

Submit your GitHub repository containing the following structure:

```
lab5/
    online-store/
        src/
        ...
```

Note: Do not push the `node_modules/` folder. Make sure your `.gitignore` excludes it. Include a `README.md` with instructions to run the project.

Useful Resources

1. **Angular Components Guide:** https://angular.dev/guide/components
2. **Component Inputs:** https://angular.dev/guide/components/inputs
3. **Component Outputs:** https://angular.dev/guide/components/outputs
4. **Component Lifecycle Hooks:** https://angular.dev/guide/components/lifecycle
5. **Angular Control Flow:** https://angular.dev/guide/templates/control-flow
6. **Angular Event Binding:** https://angular.dev/guide/templates/event-binding
7. **Angular Official Documentation:** https://angular.dev/overview
8. **TypeScript Handbook:** https://www.typescriptlang.org/docs/handbook/
9. **Kaspi.kz (for product data):** https://kaspi.kz/

GOOD LUCK! :)