COMPONENTS

Let's build an application using components, directives, and a custom directive. The project structure will look like the following. Use the command line CLI to create these files.

Part 1: Set Up Your Project Structure and Core Files

1. **Create Your Project Folder**
   Create an Angular project (here called `your-angular-app`).

2. **Build the Folder Hierarchy**
   Inside your project folder, create the following structure. Some of these files will already exist, and some will need to be created, but you can use the shortcut:

- A file named **package.json** at the root.
- A folder named **src**.
- Inside **src**, create a file named **main.ts**.
- Inside **src**, create another folder called **app**.
- Within **app**, create:
  - A file named **app.component.ts**.
  - A file named **app.component.html**.
  - A file named **book.ts**.
  - A folder named **book-inventory**.
- Inside **book-inventory**, create three files:
  - **book-inventory.component.ts**
  - **book-inventory.component.html**
  - **book-inventory.component.css**

```
your-angular-app/
├── package.json
├── src/
│   ├── main.ts
│   └── app/
│       ├── app.component.ts
│       ├── app.component.html
│       ├── book.ts
│       └── book-inventory/
│           ├── book-inventory.component.ts
│           ├── book-inventory.component.html
│           └── book-inventory.component.css
```

Part 2: Building the Root Component

1. **Creating app.component.ts**
   Open **src/app/app.component.ts**. This file represents the main component of your application.

   - You need to create an Angular component by using the `@Component` decorator.
   - **Hint:** In the decorator, specify a CSS selector (like `"app-root"`) and tell Angular that this component is “standalone.” Also, include any Angular modules that might be needed (e.g., the CommonModule) as well as any child components you plan to use—in this case, the Book Inventory component (which you'll create next).
   - In the class itself, you don’t need any special logic; this is just a container.
2. **Creating app.component.html**
   Open **src/app/app.component.html**. This file is the template for your root component.

   - **Hint:** Instead of writing content directly, you want to embed the Book Inventory component. Since that component will have its own selector (for example, `<app-book-inventory>`), simply place that tag here.

Part 3: Defining the Book Model

1. **Creating book.ts**
   Open **src/app/book.ts**. In this file, you need to define a class that represents a book.
   - **Hint:** The class should have properties like ISBN, title, author, publication year, price, whether it’s featured, and an array for cover image paths.
   - Use TypeScript’s constructor shorthand (with public parameters) so that you can automatically create and assign properties.

Part 4: Creating the Book Inventory Component

1. **Building book-inventory.component.ts**
   Open **src/app/book-inventory/book-inventory.component.ts**. This component will be responsible for displaying a list of books.

   - You need to create another standalone component here.
   - **Hint:** Use the `@Component` decorator to define the selector (perhaps `"app-book-inventory"`), set it as standalone, and list any modules (like CommonModule) that you require.
   - **Inventory Data:** Inside the component class, define a property (e.g., `inventory`) that is an array of Book objects. Hard-code a few books that

are about Angular and JavaScript. Make sure that at least one book has its featured property set to true so you can see conditional styling.
    - **Additional Functions:** Write a method to “track” each book in a loop (using its ISBN as a unique identifier) to optimize rendering, and another method to delete a book from the list.

2. **Creating book-inventory.component.html**
   Open **src/app/book-inventory/book-inventory.component.html**.

    - **Hint:** Start by placing a heading that tells the user this is the current inventory.
    - Use Angular’s structural directive `*ngIf` to display a message if the inventory is empty.
    - Use the `*ngFor` directive to loop through each book in the inventory. Inside the loop, display the book’s title, author, year, ISBN, and price. The directive should use trackBy. The code for the trackBy function in the .ts file could look like this: `trackByISBN(index: number, book: Book): string { return book.ISBN; }`
    - **Conditional Display:** Use `*ngIf` to show a special message for featured books.
    - **ngSwitch:** Add a section where you use the `[ngSwitch]` directive to show different messages based on the price of the book. For example, if the price is greater than 30, show “This is a premium selection!”; if not, show “A budget-friendly choice!” or a default message.
    - Finally, include a button that, when clicked, calls your delete method to remove that book from the inventory.
    - The html component could look like the following:

```
<div *ngFor="let book of inventory; trackBy: trackByISBN"
     [ngClass]="{'featured-item': book.featured}"
     appHoverHighlight=“#e1e1e1">
  <!-- Rest of the book template remains the same -->
  <h3>{{book.title}} by {{book.author}} ({{book.year}})</h3>
  <p *ngIf="book.featured">Featured book of the month!</p>
  <p>ISBN: {{book.ISBN}}</p>
  <p>Price: ${{book.price}}</p>

  <div [ngSwitch]="book.price > 30 ? 'expensive' :
'affordable'">
    <p *ngSwitchCase="'expensive'">This is a premium selection!
</p>
    <p *ngSwitchCase="'affordable'">A budget-friendly choice!</
p>
    <p *ngSwitchDefault>Great value!</p>
  </div>
```

```
  <p>
    <button (click)="deleteBook(book)">Delete</button>
  </p>
</div>
```

3. **Optional: install Bootstrap, configure the json files, and add Bootstrap classes. The code might look like this:**

```
<div class="container my-4">

    <h1 class="mb-4 text-center">Current Inventory</h1>

    <p *ngIf="inventory.length === 0" class="alert alert-info
text-center">There are no books in inventory.</p>

    <div class="row">

      <!-- Each book occupies a responsive column: full width on
extra small screens, half on small, one-third on medium, and
one-fourth on large screens -->

      <div class="col-12 col-sm-6 col-md-4 col-lg-3 mb-4"
*ngFor="let book of inventory; trackBy: trackByISBN">

        <!-- Use h-100 to enforce equal height and add a subtle
shadow -->

        <div class="card h-100 shadow-sm"
appHoverHighlight="#e1e1e1">

            <div class="ratio ratio-4x3">

                <img *ngIf="book.coverImages.length > 0"

                     [src]="book.coverImages[0]"

                     class="card-img-top"

                     alt="{{ book.title }} cover"

                     style="object-fit: cover;">

              </div>

            <div class="card-body d-flex flex-column">

            <h5 class="card-title">{{ book.title }}</h5>
```

```
            <p class="card-text flex-grow-1">
              <strong>Author:</strong> {{ book.author }}<br>
              <strong>Year:</strong> {{ book.year }}<br>
              <strong>ISBN:</strong> {{ book.ISBN }}<br>
              <strong>Price:</strong> ${{ book.price }}
            </p>
            <!-- Featured badge -->
            <p *ngIf="book.featured" class="badge bg-primary
mb-2">Featured Book</p>
            <!-- ngSwitch for price-category message -->
            <div [ngSwitch]="book.price > 30 ? 'expensive' :
'affordable'" class="mb-2">
              <p *ngSwitchCase="'expensive'" class="text-danger
mb-0">Premium selection!</p>
              <p *ngSwitchCase="'affordable'" class="text-
success mb-0">Budget-friendly!</p>
              <p *ngSwitchDefault class="text-muted mb-0">Great
value!</p>
            </div>
            <!-- Delete button stays at the bottom using mt-auto
-->
            <button (click)="deleteBook(book)" class="btn btn-
danger mt-auto">Delete</button>
          </div>
        </div>
      </div>
    </div>
  </div>
```

4. **Creating book-inventory.component.css**
   Open **src/app/book-inventory/book-inventory.component.css**.

   - **Hint:** Write a CSS rule that styles a “featured” book differently. For example, add a border, some padding, rounded corners, and perhaps a different background color. Name the CSS class something like `.featured-item`.

Part 6: Adding a Custom Attribute Directive

1. **Creating the Directive File**
   In the **book-inventory** folder, create a new file called **hover-highlight.directive.ts**.

   - **Hint:** You’re creating a custom directive that changes an element’s background color when the mouse enters and reverts it when the mouse leaves.
   - Use the `@Directive` decorator. In the selector, use an attribute format (for example, `[appHoverHighlight]`).
   - In your directive class, inject an `ElementRef` so that you can manipulate the DOM element.
   - Use `@HostListener` to listen for `mouseenter` and `mouseleave` events.
   - The directive should have an input property that allows you to set the highlight color.
   - Write the functions you need so that when the mouse enters, the current background color is stored and then replaced with your chosen highlight color, and when the mouse leaves, the original color is restored.
2. **Integrate the Directive into the Component**
   Open **book-inventory.component.ts** and import your new directive. Then add it to the component’s `imports` array in the decorator.

   - **Hint:** This lets you use your custom directive in the component’s template.
3. **Use the Directive in the Template**
   In **book-inventory.component.html**, add your directive attribute (for example, `appHoverHighlight="#e1e1e1"`) to the outer `<div>` that represents each book item.

   - **Hint:** Now, when you hover over a book item, its background should change to the specified color (light gray).
   - The custom direct code will be something like this. There is code to account for a smooth transition between colors:

```
// src/app/book-inventory/hover-highlight.directive.ts
import { Directive, ElementRef, HostListener, Input } from
'@angular/core';

@Directive({
  selector: '[appHoverHighlight]',
  standalone: true
})
export class HoverHighlightDirective {
  @Input('appHoverHighlight') highlightColor: string = '';
  private defaultColor: string = '';

  constructor(private el: ElementRef) {
    // Set a transition on the background-color for smooth
animations.
    this.el.nativeElement.style.transition = 'background-color
0.3s ease';
  }

  @HostListener('mouseenter')
  onMouseEnter(): void {
    // Store the current background color, then change it to the
highlight color.
    this.defaultColor =
this.el.nativeElement.style.backgroundColor;
    this.highlight(this.highlightColor);
  }

  @HostListener('mouseleave')
  onMouseLeave(): void {
    // Revert back to the original background color.
    this.highlight(this.defaultColor);
  }

  private highlight(color: string): void {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```