Task: Implement State Management with a Service in Angular

Objective

Create an Angular application that demonstrates state management using a service. The application should have a shared state (a counter) that can be displayed and modified from different components.

Requirements

1. Service Creation
   - Create a service named `CounterService` using Angular CLI (`ng generate service counter`).
   - Use a `BehaviorSubject` from RxJS to manage the counter state, initialized to 0.
   - Provide public methods:
      - `increment()`: Increases the counter by 1.
      - `decrement()`: Decreases the counter by 1.
      - `reset()`: Resets the counter to 0.
   - Expose the counter state as a public observable (e.g., `count$`) that components can subscribe to, derived from the `BehaviorSubject`.

2. Component Creation
   - Create two components using Angular CLI:
      - `CounterDisplayComponent` (`ng generate component counter-display`): Displays the current counter value.
      - `CounterControlComponent` (`ng generate component counter-control`): Provides buttons to modify the counter.

3. Component Implementation
   - CounterDisplayComponent:
      - Inject the `CounterService`.
      - Use the async pipe in the template to display the current counter value from the service's observable (e.g., `{{ counterService.count$ | async }}`).
   - CounterControlComponent:
      - Inject the `CounterService`.
      - Include three buttons in the template: "Increment", "Decrement", and "Reset".
      - Bind the buttons' `(click)` events to call the corresponding methods (`increment()`, `decrement()`, `reset()`) from the `CounterService`.

4. Integration in AppComponent
   - In the `AppComponent` template, include at least one instance of each component:

```
<app-counter-display></app-counter-display>
<app-counter-control></app-counter-control>
```

- Ensure that clicking the buttons in `CounterControlComponent` updates the counter value displayed in `CounterDisplayComponent`.

5. Best Practices
- In the `CounterService`, use the `@Injectable({ providedIn: 'root' })` decorator to make it a singleton available throughout the application.
- Leverage the async pipe in `CounterDisplayComponent` to automatically manage subscription and unsubscription to the observable.

Optional Enhancements
- Add multiple `<app-counter-display>` instances in the `AppComponent` template to demonstrate that all instances reflect the same shared state.
- Modify the `decrement()` method to prevent the counter from going below zero (e.g., only decrease if the current value is greater than 0).

Deliverables
- The complete source code of the Angular application, including the service and components.
- A brief explanation (e.g., in a README file or comments) describing how the state is managed and shared between components using the service.

Additional Notes
- Ensure all necessary imports (e.g., `BehaviorSubject` from `'rxjs'`) are included.
- The application should run without errors, and the counter state should be correctly shared and updated across components.
- Assume a standard Angular setup with Angular CLI; no additional state management libraries (like NgRx) are required for this task.