Task: Implement a Counter Application Using Signals in Angular 19

Objective

Create an Angular 19 application that uses signals to manage a shared counter state across components.

Requirements

1. CounterService
   - Create a service named `CounterService`.
   - Define a signal `count` initialized to `0`.
   - Define a computed signal `isEven` that returns `true` if `count` is even, `false` otherwise.
   - Implement methods:
      - `increment()`: Increases `count` by 1.
      - `decrement()`: Decreases `count` by 1, but not below 0.
      - `reset()`: Sets `count` to 0.
   - Expose `count` as a read-only signal.

2. CounterDisplayComponent
   - Create a standalone component named `CounterDisplayComponent`.
   - Inject `CounterService` using `inject()`.
   - Display the current `count` value and `isEven` status in the template (e.g., "Counter: 4 (Even)").

3. CounterControlsComponent
   - Create a standalone component named `CounterControlsComponent`.
   - Inject `CounterService` using `inject()`.
   - Include three buttons in the template:
      - "Increment": Calls `increment()`.
      - "Decrement": Calls `decrement()`.
      - "Reset": Calls `reset()`.

4. AppComponent
   - In the standalone `AppComponent`, include:
      - At least two instances of `CounterDisplayComponent`.
      - One instance of `CounterControlsComponent`.
   - Ensure updates from `CounterControlsComponent` reflect in all `CounterDisplayComponent` instances.

Notes

- Use `@Injectable({ providedIn: 'root' })` for `CounterService`.
- Access signal values in templates with function call syntax (e.g., `{{ count() }}`).
- Ensure all components are standalone and properly import required dependencies.