**Front-end architecture challenge Overview:**

You have four mock images depicting a dashboards page:

1. **Dashboard Overview**: Displays multiple charts (line chart, bar chart, pie charts) arranged in a grid layout.
2. **Filter Panel**: A collapsible side panel where the user can build filter clauses (e.g., “Gender includes [Male, Female] AND Age > 15”).
3. **Dashboard Selector**: A dropdown containing saved dashboards, plus an option to create a new one.
4. **Chart Configuration**: A “Chart Settings” panel where the user chooses fields for X-axis, Y-axis, and an aggregation (“Count,” “Distinct Count,” “Average,” etc.).

Using these reference images, **design a front-end application architecture** that enables:

* Multiple dashboards, each containing a set of visualizations.
* A flexible filters panel to apply constraints to the dashboard data.
* A “Chart Settings” mechanism for each visualization that determines what data is shown and how it’s aggregated.

**Important**: You are **not** being asked to replicate the exact UI or visual design. Instead, show how you’d **structure** the application’s **front-end code**, **components**, **state management**, **data flow**, and **extensibility** to support these features.  
  
**Requirements**

1. **Multiple Dashboards**
   * Users can pick a dashboard from a dropdown. Each dashboard can have several chart “widgets” displayed.
   * Users can create a new dashboard and give it a title.
2. **Filters Panel**
   * A side panel allows the user to define multiple clauses (e.g., “Gender = X,” “Age > Y”) with logical AND.
   * Applied filters affect **all** charts within the active dashboard.
3. **Chart Configuration**
   * Each chart is configured with an X‐axis field, Y‐axis field, and an aggregation method (Count, Distinct Count, etc.).
   * Users can also give the chart a custom name or description.
4. **Data Model and State Management**
   * Show how you would structure the data for dashboards, charts, filters, etc.
   * Demonstrate how the front-end fetches data and applies filters.
   * Consider using a global store (e.g., Redux, NgRx, Vuex) or a smaller library (RxJS BehaviorSubject, Context API, etc.) that best fits your chosen framework.
5. **Technology Choice**
   * Detail modules, services, and any relevant Nx/Ngrx patterns.
   * Provide a basic folder structure to illustrate how you would organize modules, components, services, store slices, etc.
6. **Extensibility**
   * The design should allow new chart types (e.g., line, bar, pie) to be added without major refactoring.
   * The filter logic should be easily extendable to more fields or different operators (e.g., “<,” “!=,” “contains,” etc.).
7. **Mock Data**
   * You can assume an API endpoint (e.g., GET /api/dashboardData) returns mock data. You only need to demonstrate how your front end might call it and apply transformations/filters.
   * Hardcode or mock the data in your solution if you prefer, but show how it **would** integrate with a real endpoint.
8. **Documentation**
   * Summarize your approach to key topics like “Where is state stored?”, “How do charts subscribe to filter changes?”, “How do we handle multiple dashboards at once?”.
   * Architecture/Design Explanation:
     + A short write-up (or inline doc) explaining your core component hierarchy, data flow, and how you manage states (dashboards, chart definitions, filters). This is where you demonstrate your architectural decisions.
     + Optional but encouraged: Provide a sample or partial code base that includes the basic scaffolding—e.g., a single page with a simple “Dashboard” component, a “FiltersPanel” component, and a “ChartWidget” component.
     + Show at least one example of how you’d store or fetch data, apply a filter, and display it in a chart. (Charts can be dummy placeholders if you don’t want to integrate a real chart library.)